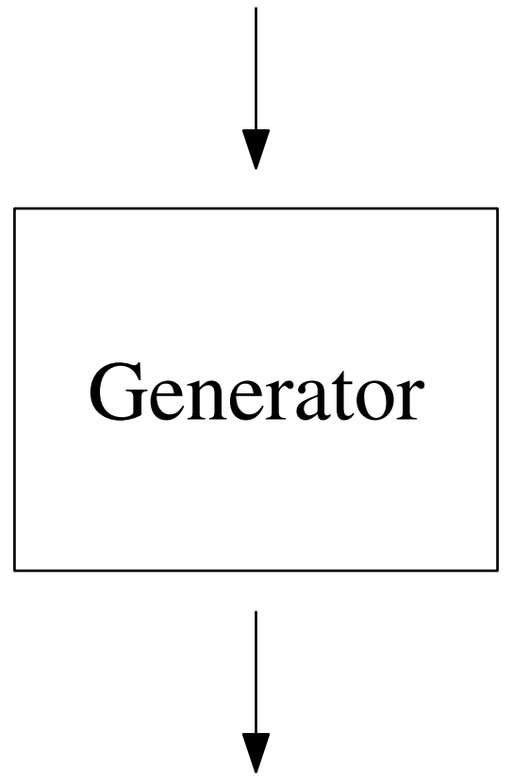
Generating k-independent variables in constant time

Tobias Christiani, Rasmus Pagh

IT University of Copenhagen

Random seed

11010100 01101101



Pseudorandom output sequence

00011011 11111100 10010101 11010001 11100110

k-independent variables

Generator output:

- Sequence of k-independent variables over a finite field $\mathbb{F}$

k-independent variables

Generator output:

- Sequence of k-independent variables over a finite field $\mathbb{F}$

Example field:

- The set of integers modulo a prime $\mathbb{F}_p$

Y_1	Y_2	Y_3	. . .	Y_p

k-independent variables

Generator output:

- Sequence of k-independent variables over a finite field $\mathbb{F}$

Example field:

- The set of integers modulo a prime $\mathbb{F}_p$

Y_1	Y_2	Y_3	. . .	Y_p

- k-independence:
  - Y_i uniformly distributed over $\mathbb{F}_p$
  - The variables at every set of k distinct positions are independent

$$\Pr[Y_{i_1} = y_1, Y_{i_2} = y_2, \ldots, Y_{i_k} = y_k] = p^{-k}$$

k-independent variables

Generator output:

- Sequence of k-independent variables over a finite field $\mathbb{F}$

Example field:

- The set of integers modulo a prime $\mathbb{F}_p$

Y_1	Y_2	Y_3	. . .	Y_p

- k-independence:
  - Y_i uniformly distributed over $\mathbb{F}_p$
  - The variables at every set of k distinct positions are independent

$$\Pr[Y_{i_1} = y_1, Y_{i_2} = y_2, \ldots, Y_{i_k} = y_k] = p^{-k}$$

- Many properties of full randomness for $k \ll p$

Model and problem

Model:

- Word-RAM model
  - Elements of $\mathbb{F}_p$ fit in a single word
  - Constant time addition and multiplication in $\mathbb{F}_p$

Model and problem

Model:

- Word-RAM model
  - Elements of $\mathbb{F}_p$ fit in a single word
  - Constant time addition and multiplication in $\mathbb{F}_p$

Problem:

- Construct a data structure that outputs a k-independent sequence over $\mathbb{F}_p$, one variable at the time
  - Low time complexity of generating next element
  - Space usage and initialization time close to k

Main result

Explicit data structure for generating the next element of a k-independent sequence in worst case *constant time* using space and initialization time $k \operatorname{poly} \log k$

Main result

Explicit data structure for generating the next element of a k-independent sequence in worst case *constant time* using space and initialization time $k \operatorname{poly} \log k$

- Constant time generation for every choice of k almost as large as p

Main result

Explicit data structure for generating the next element of a k-independent sequence in worst case *constant time* using space and initialization time $k \operatorname{poly} \log k$

- Constant time generation for every choice of k almost as large as p
- The generator always outputs a k-independent sequence

Main result

Explicit data structure for generating the next element of a k-independent sequence in worst case *constant time* using space and initialization time $k \operatorname{poly} \log k$

- Constant time generation for every choice of k almost as large as p
- The generator always outputs a k-independent sequence
- Space in bits: $(k \operatorname{poly} \log k) \cdot \log p$
  - Logarithmic dependence on length of output sequence

Main result

Explicit data structure for generating the next element of a k-independent sequence in worst case *constant time* using space and initialization time $k \operatorname{poly} \log k$

- Constant time generation for every choice of k almost as large as p
- The generator always outputs a k-independent sequence
- Space in bits: $(k \operatorname{poly} \log k) \cdot \log p$
  - Logarithmic dependence on length of output sequence

COMPARISON WITH PREVIOUS RESULTS

Reference	**Space**	**Generation time**
Multipoint evaluation [GG'13]	$O(k \log k)$	$O(\log^2 k \log \log k)$
Expander hashing [Siegel'89]	$O(p^\varepsilon),\ \varepsilon > 0$	$O(1)$
Our generator	$O(k \operatorname{poly} \log k)$	$O(1)$

k-independent hash functions

- Data structure for *random access* to a k-independent sequence

k-independent hash functions

- Data structure for *random access* to a k-independent sequence

Polynomial hash functions [Joffe'74, CW'81]

- Choose k random elements from $\mathbb{F}_p$ and use them as coefficients of a polynomial

$$h(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} \bmod p$$

k-independent hash functions

- Data structure for *random access* to a k-independent sequence

Polynomial hash functions [Joffe'74, CW'81]

- Choose k random elements from $\mathbb{F}_p$ and use them as coefficients of a polynomial

$$h(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} \bmod p$$

- Polynomial hash function as a generator
  - Space k to store coefficients
  - Evaluation time $O(k)$

$h(0)$	$h(1)$	$h(2)$	$\cdot \quad \cdot \quad \cdot$	$h(p-1)$

- Space usage near-optimal with respect to sequence length

k-independent hashing: lower bound

- What is the best time-space tradeoff we can hope for using hashing?

k-independent hashing: lower bound

- What is the best time-space tradeoff we can hope for using hashing?
- Cell probe lower bound [Siegel'89]

$$k\text{-independent hash function } h : \mathbb{F}_p \to \mathbb{F}_p$$

$$\text{Time } t < k \Rightarrow$$
$$\text{Space at least } p^{1/t} \text{ words}$$

k-independent hashing: lower bound

- What is the best time-space tradeoff we can hope for using hashing?

- Cell probe lower bound [Siegel'89]

$$k\text{-independent hash function } h : \mathbb{F}_p \to \mathbb{F}_p$$

$$\text{Time } t < k \Rightarrow$$
$$\text{Space at least } p^{1/t} \text{ words}$$

- Constructions in the word-RAM model still far from lower bound

k-generators

- Data structure for *sequential access* to a k-independent sequence

k-generators

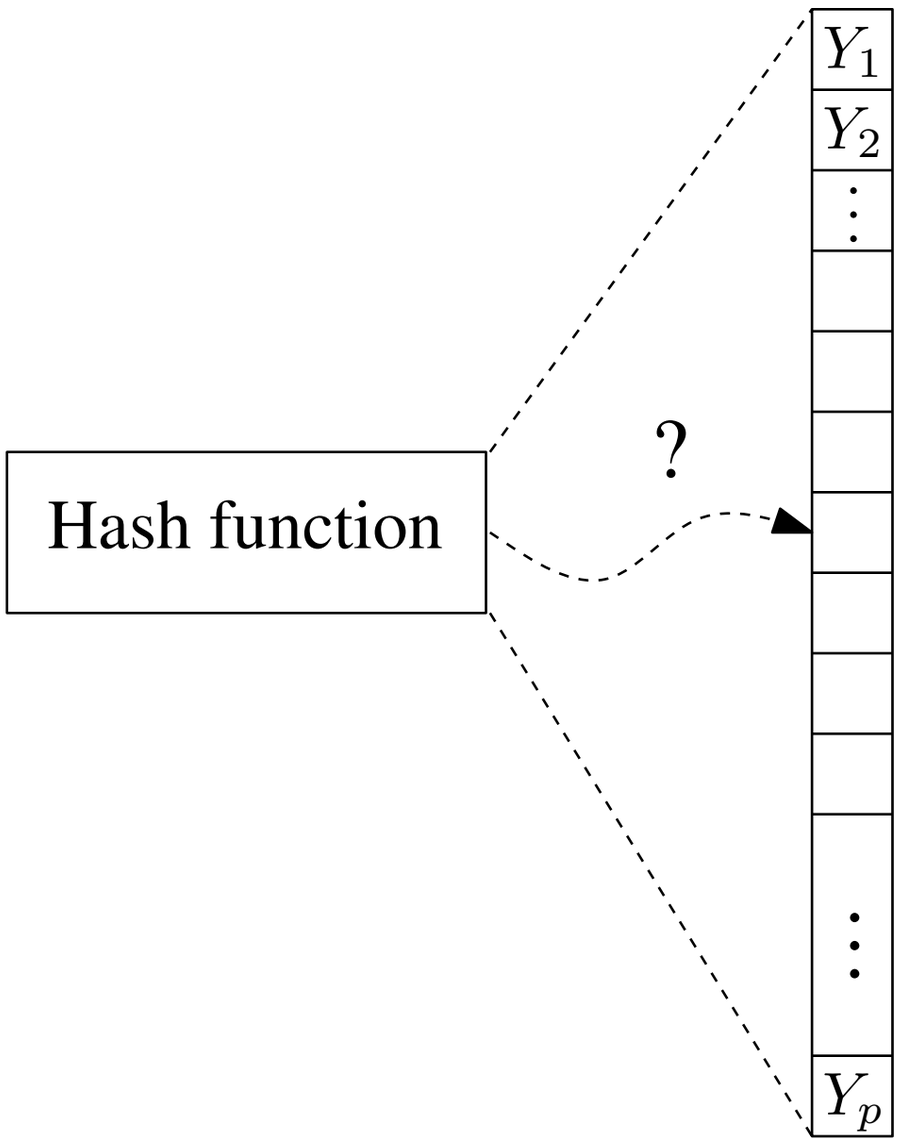
- Data structure for *sequential access* to a k-independent sequence
- Circumvent hashing lower bound

k-generators

- Data structure for *sequential access* to a k-independent sequence
- Circumvent hashing lower bound
- Sequence of local hash functions evaluated over small domains

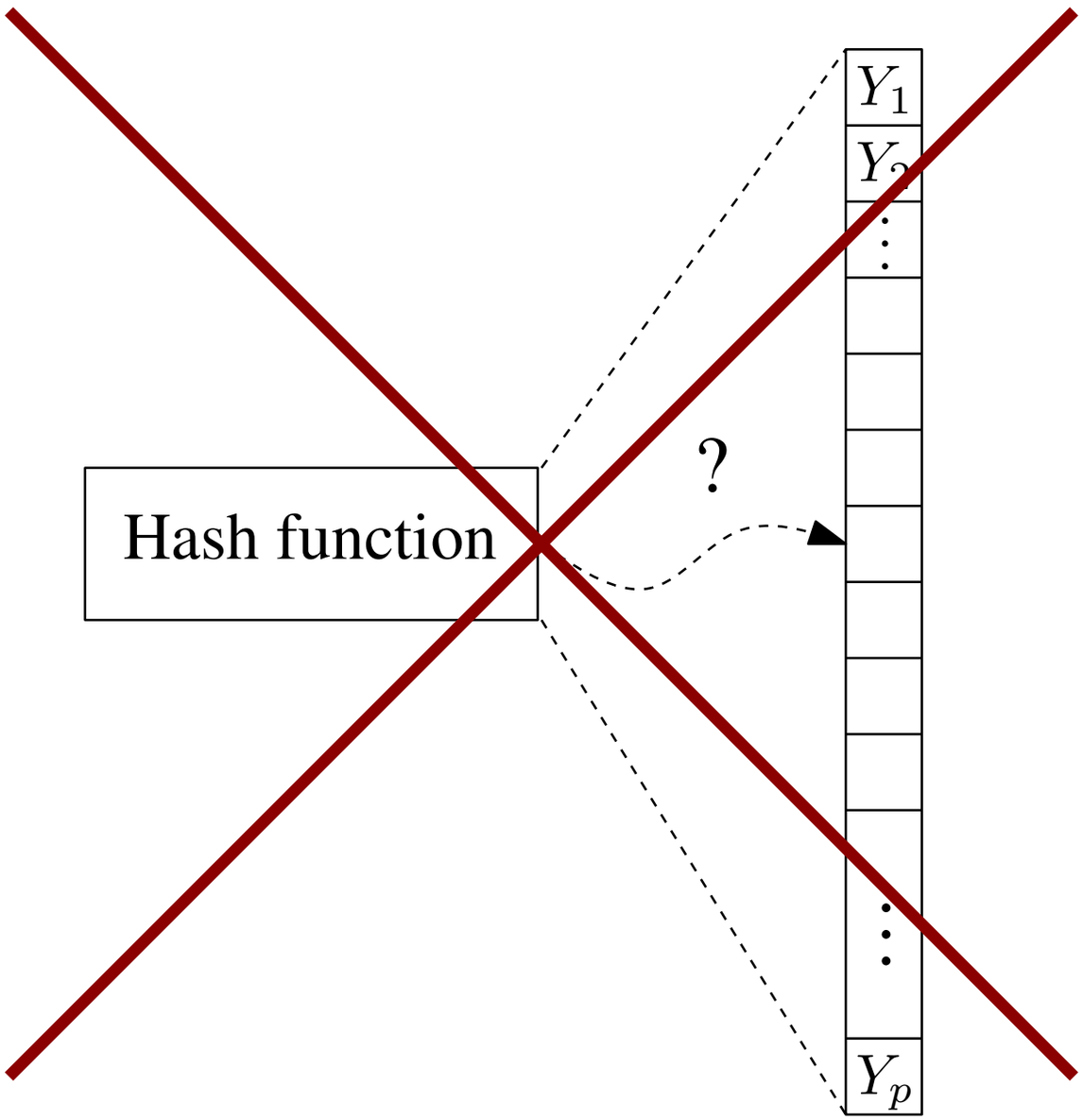
k-generators

- Data structure for *sequential access* to a k-independent sequence
- Circumvent hashing lower bound
- Sequence of local hash functions evaluated over small domains



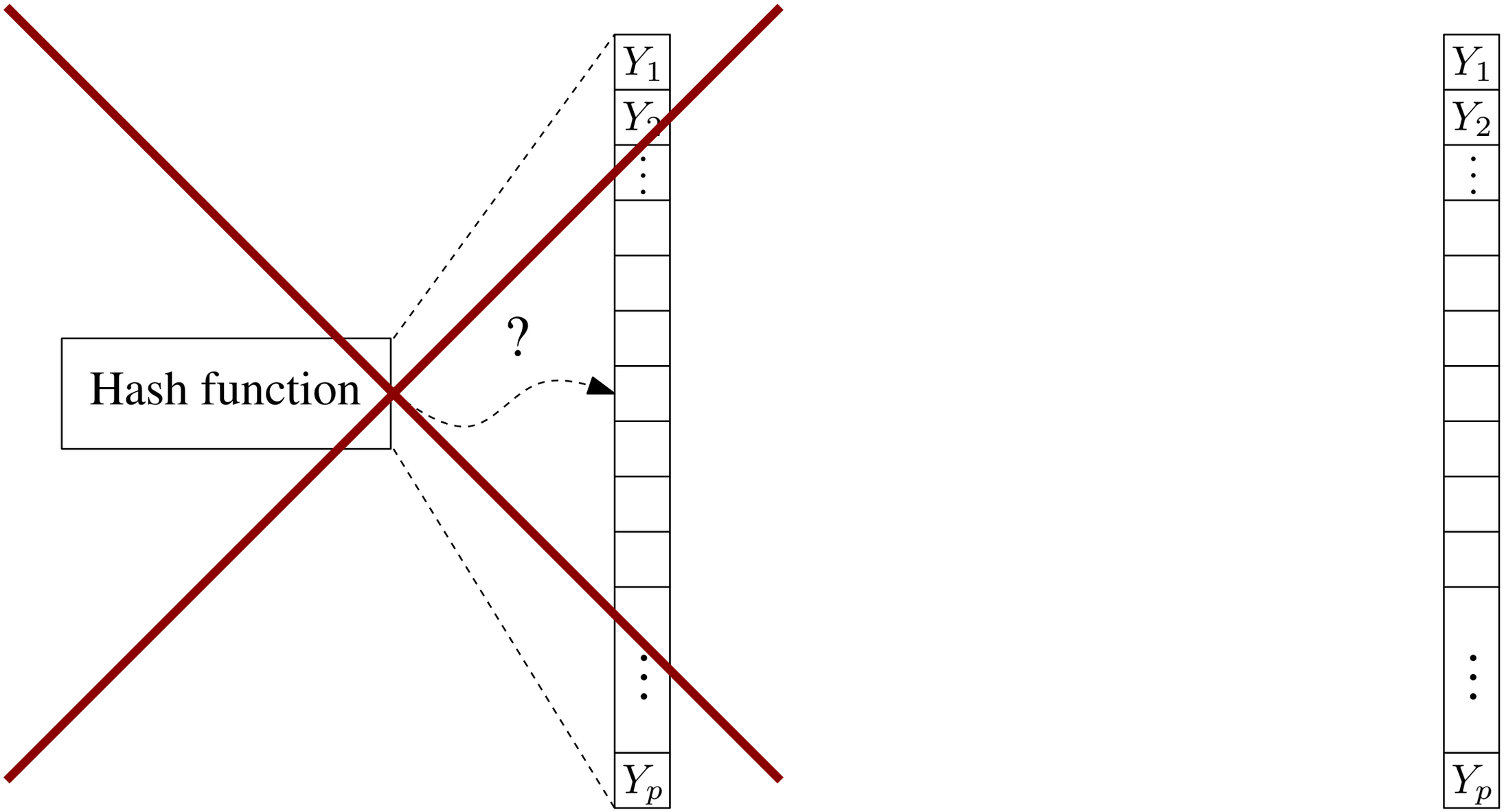
k-generators

- Data structure for *sequential access* to a k-independent sequence
- Circumvent hashing lower bound
- Sequence of local hash functions evaluated over small domains



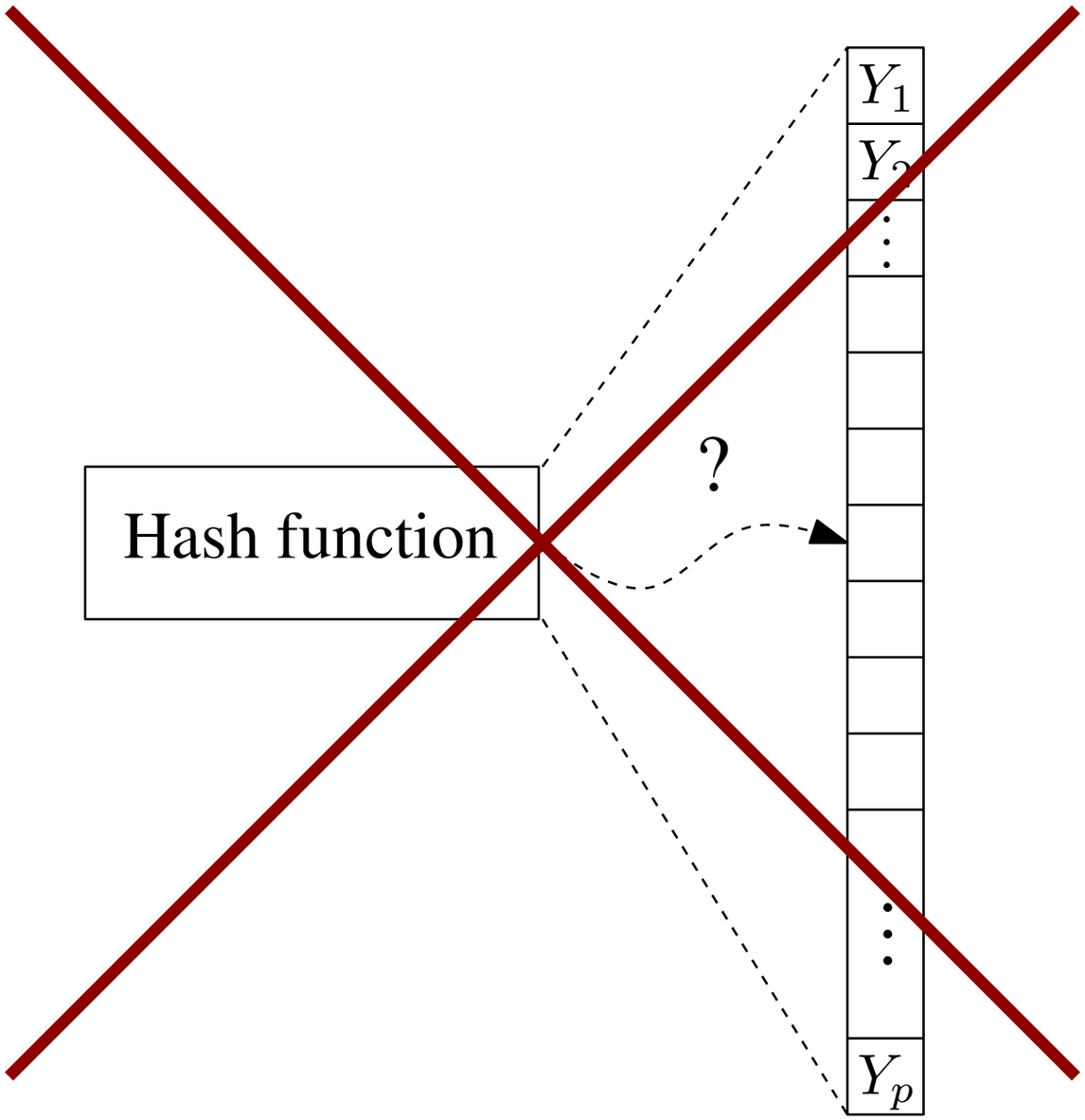
k-generators

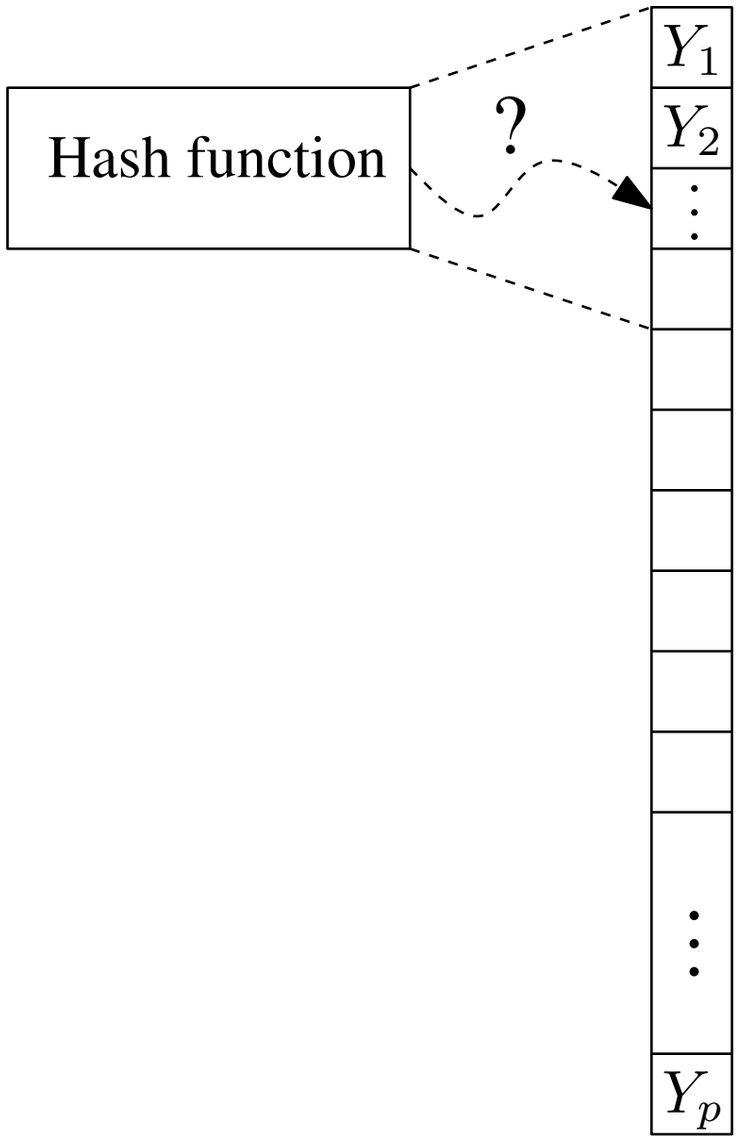
- Data structure for *sequential access* to a k-independent sequence
- Circumvent hashing lower bound
- Sequence of local hash functions evaluated over small domains



k-generators

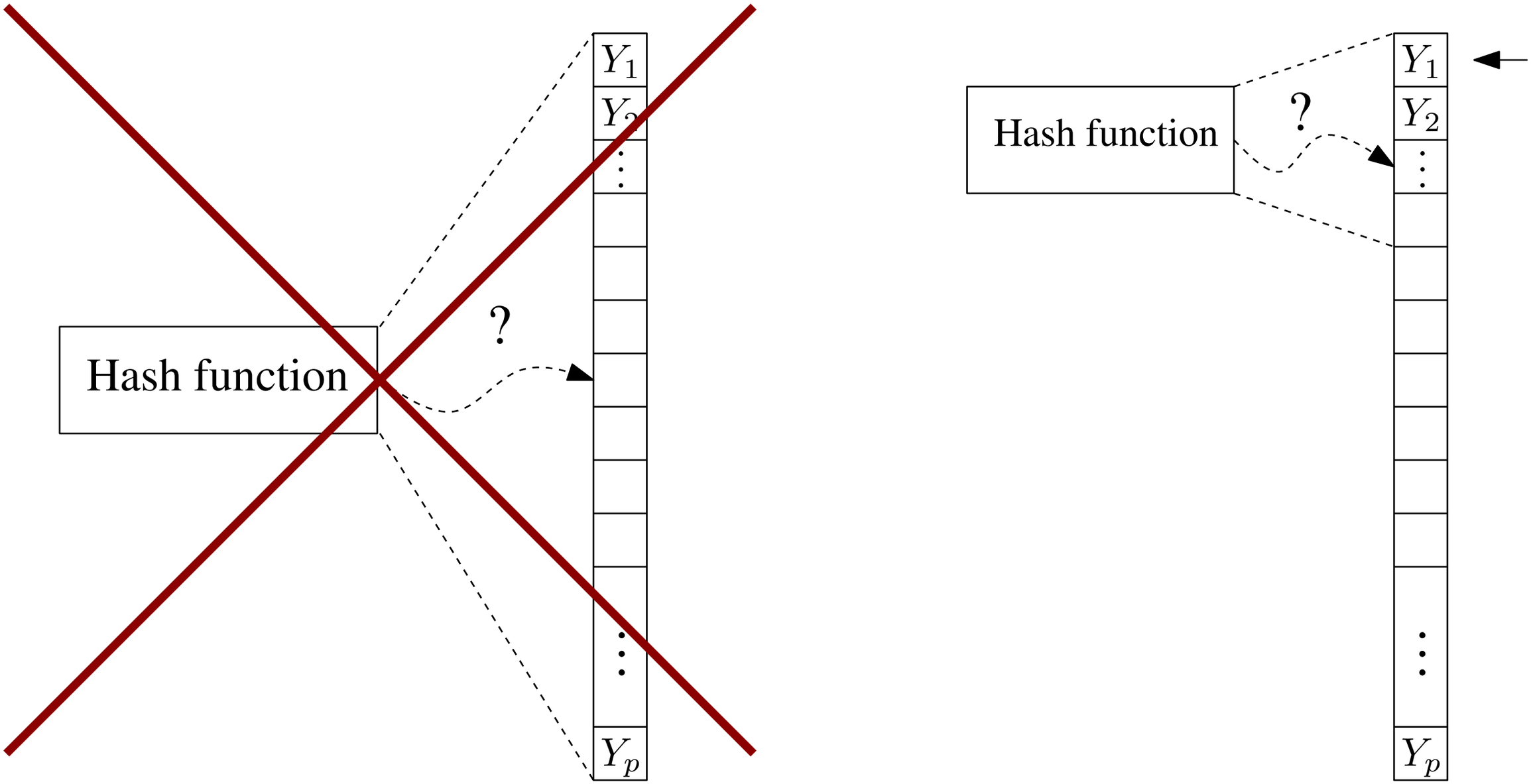
- Data structure for *sequential access* to a k-independent sequence
- Circumvent hashing lower bound
- Sequence of local hash functions evaluated over small domains





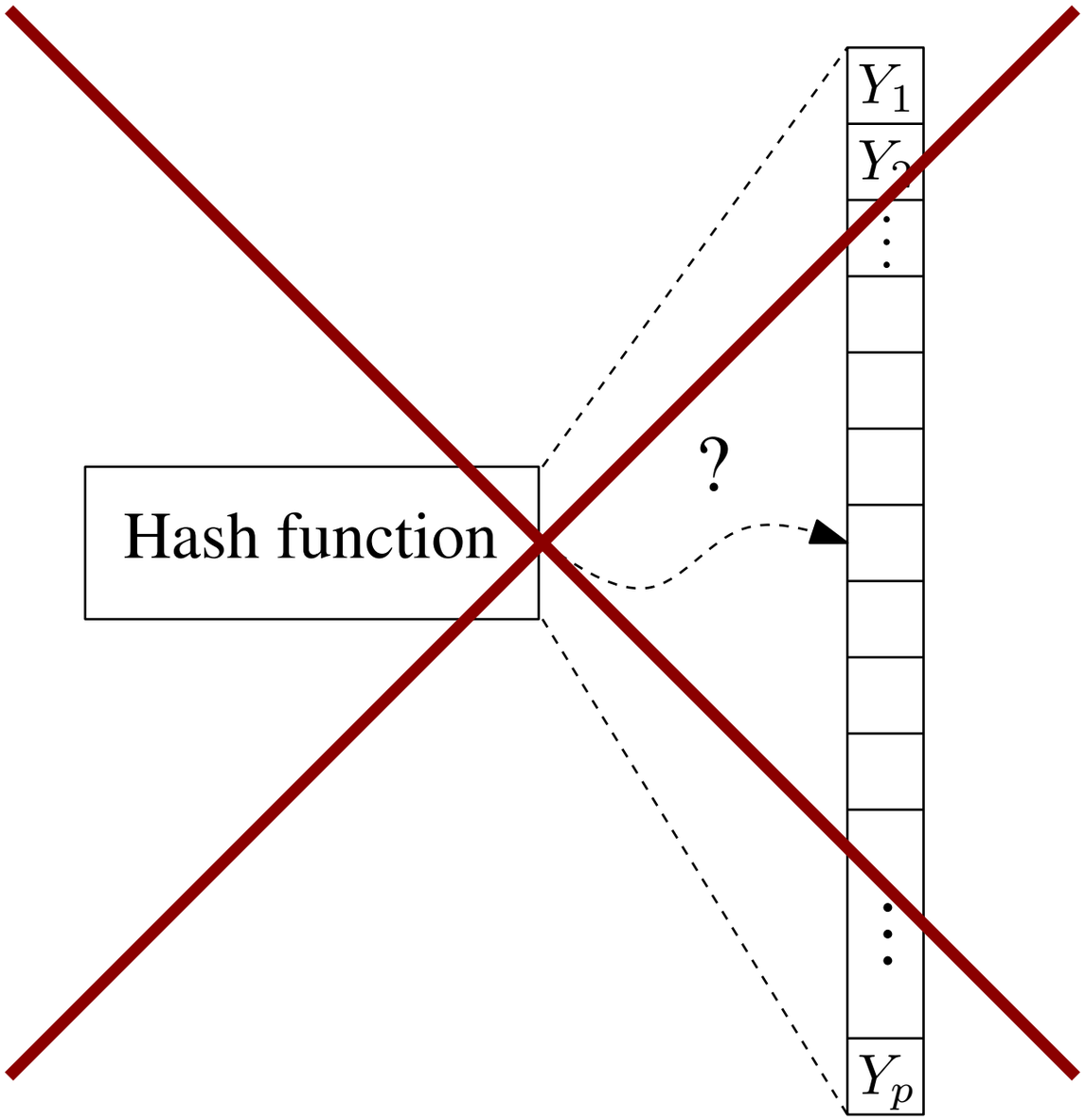
k-generators

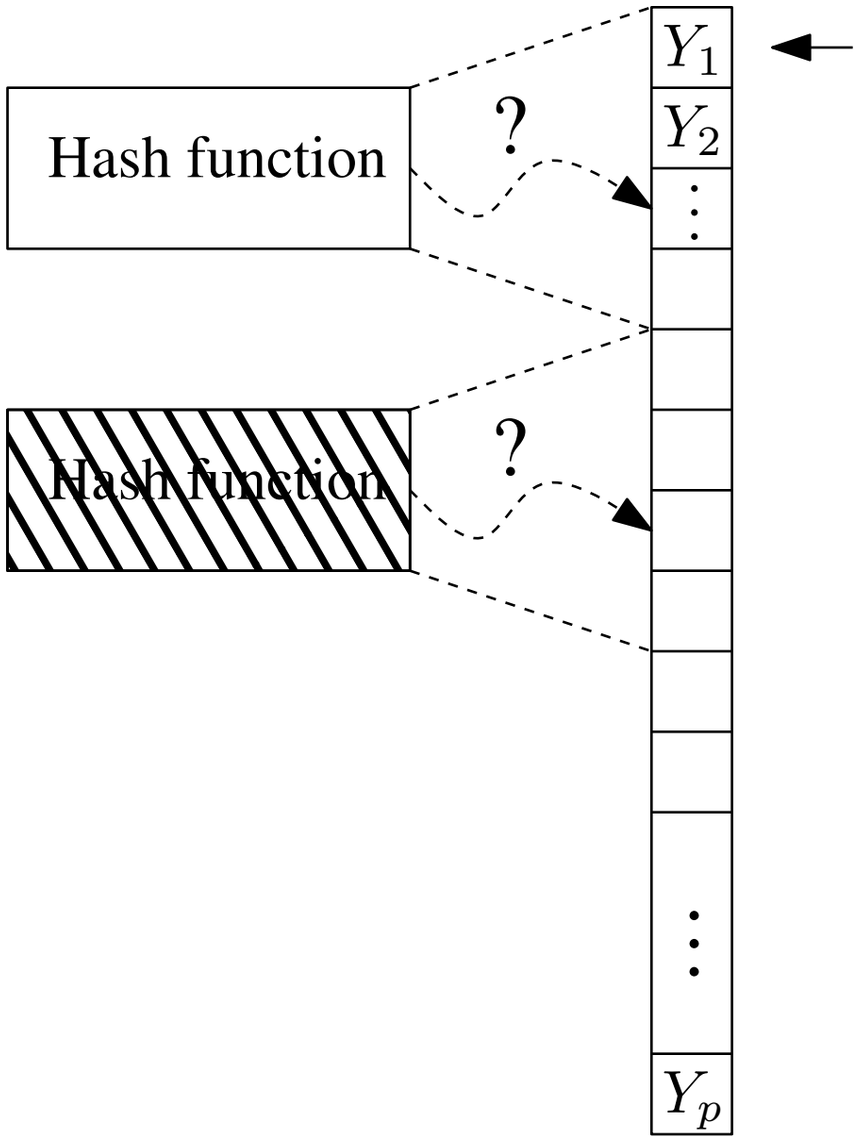
- Data structure for *sequential access* to a k-independent sequence
- Circumvent hashing lower bound
- Sequence of local hash functions evaluated over small domains



k-generators

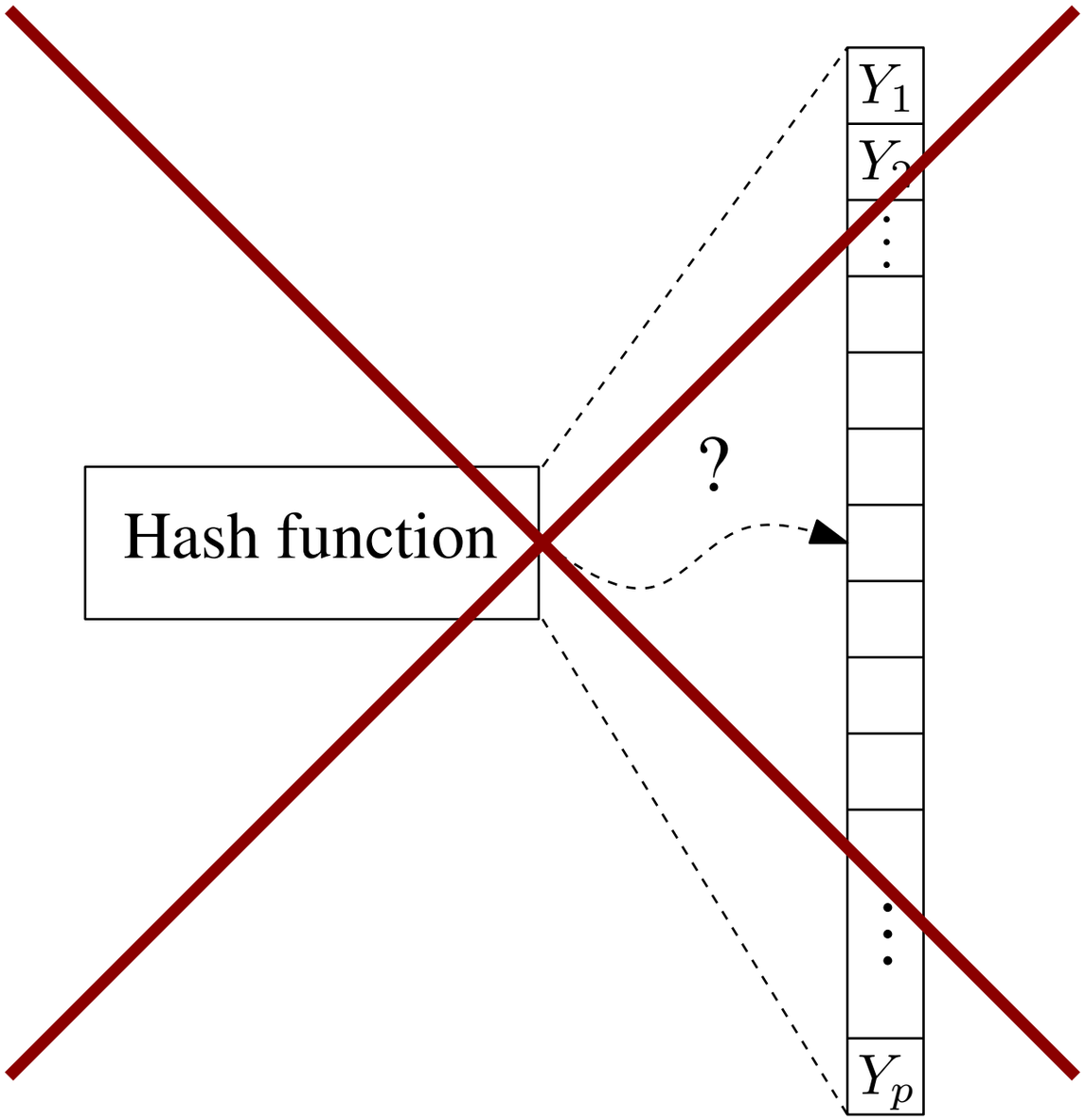
- Data structure for *sequential access* to a k-independent sequence
- Circumvent hashing lower bound
- Sequence of local hash functions evaluated over small domains

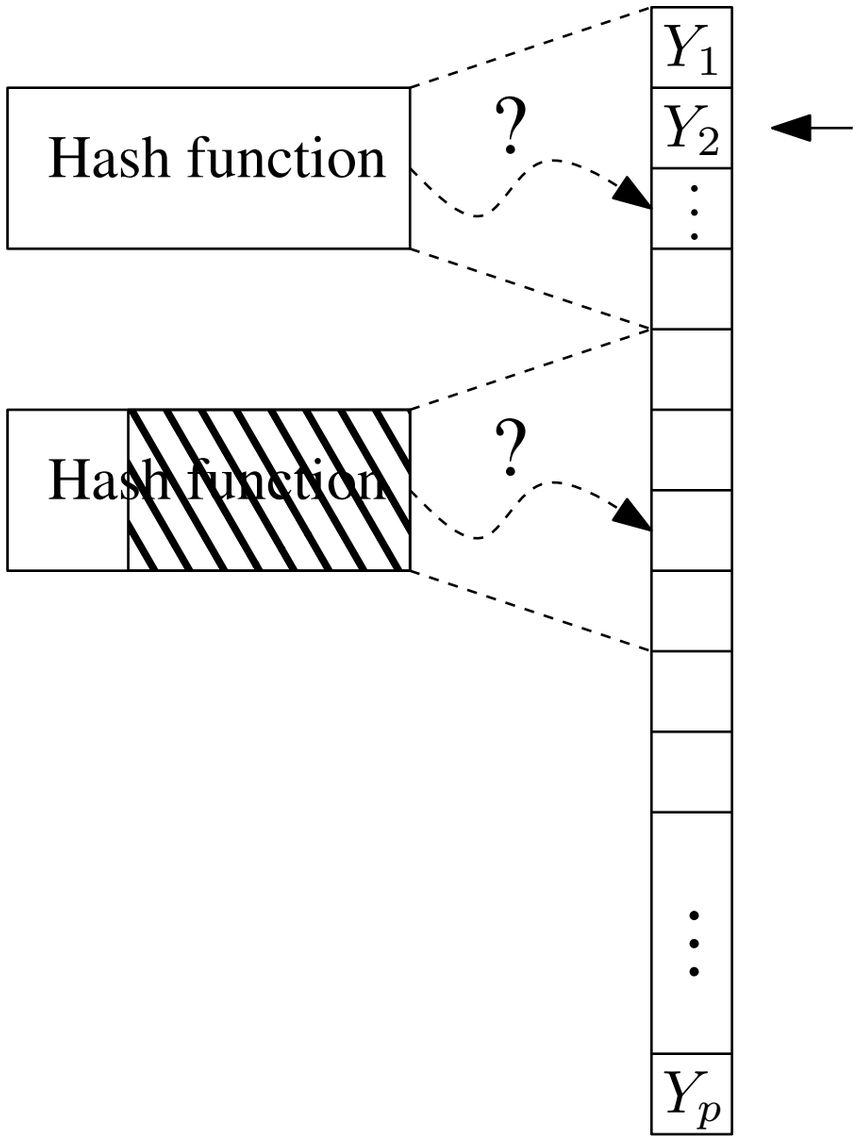




k-generators

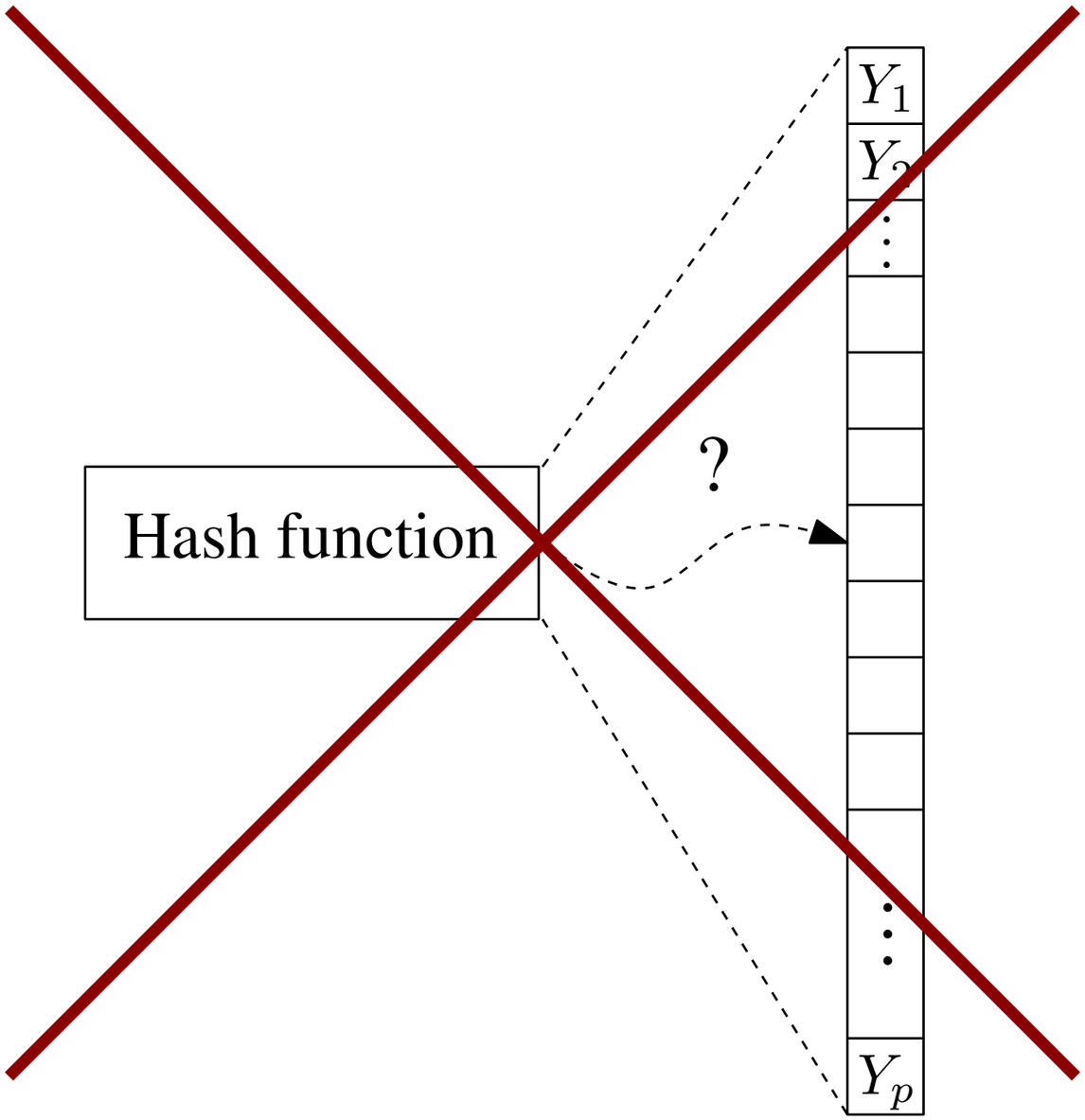
- Data structure for *sequential access* to a k-independent sequence
- Circumvent hashing lower bound
- Sequence of local hash functions evaluated over small domains

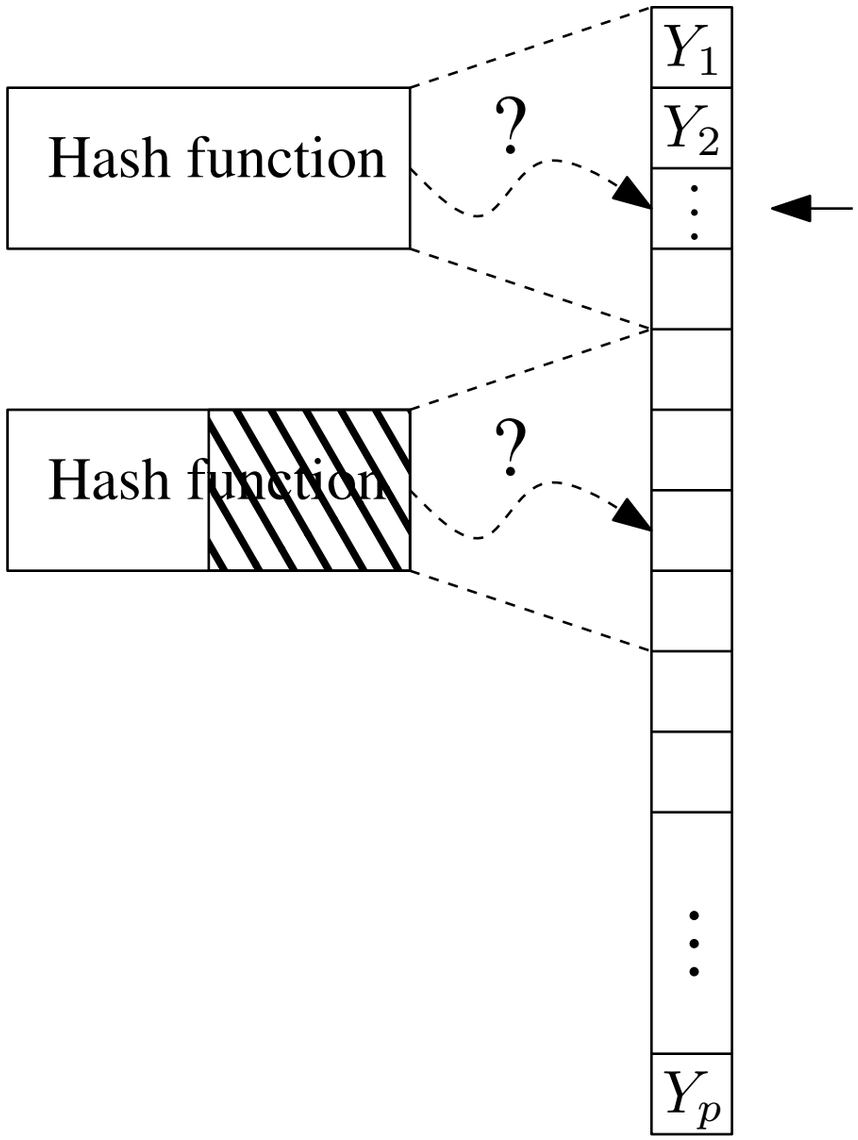




k-generators

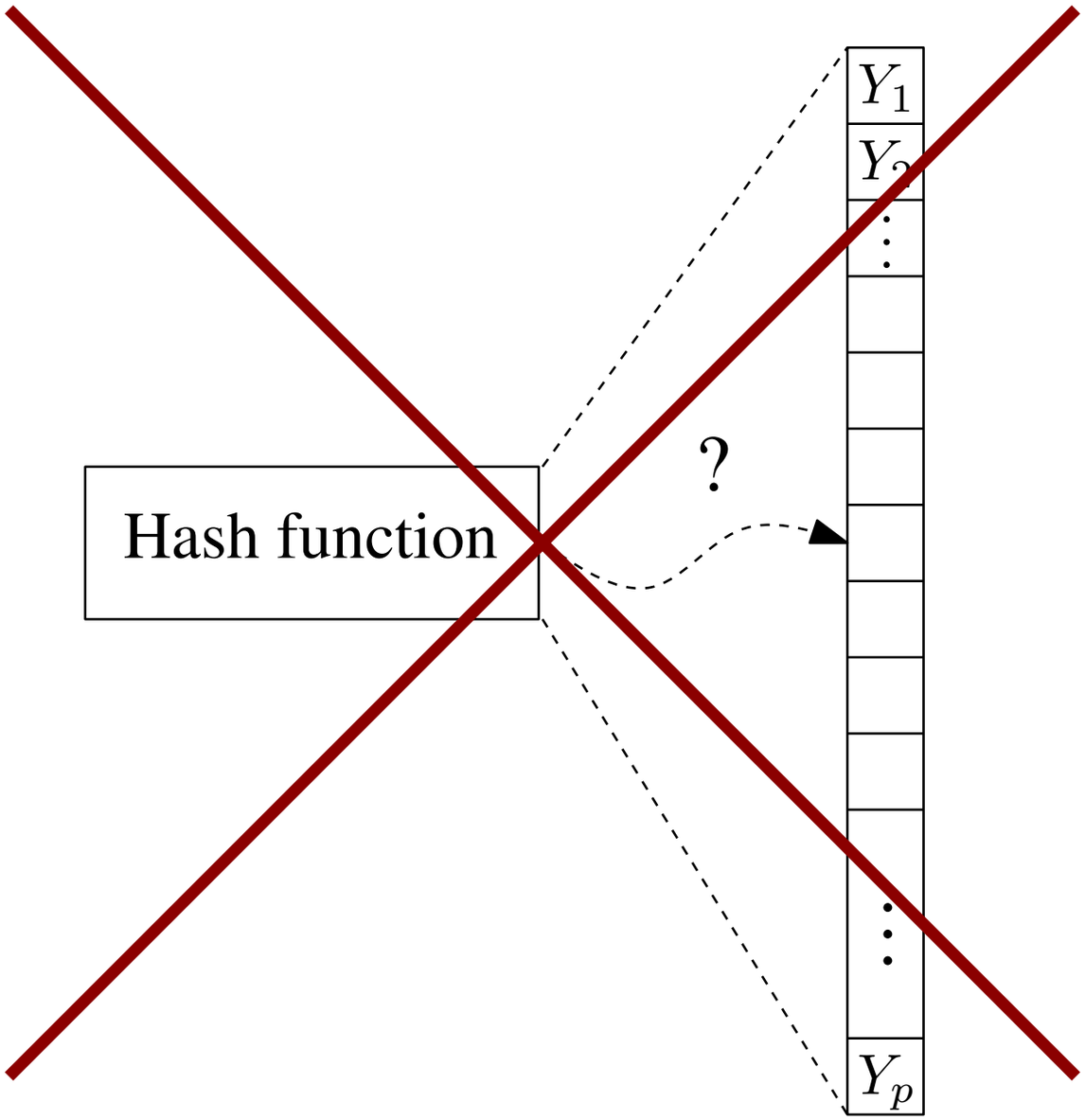
- Data structure for *sequential access* to a k-independent sequence
- Circumvent hashing lower bound
- Sequence of local hash functions evaluated over small domains

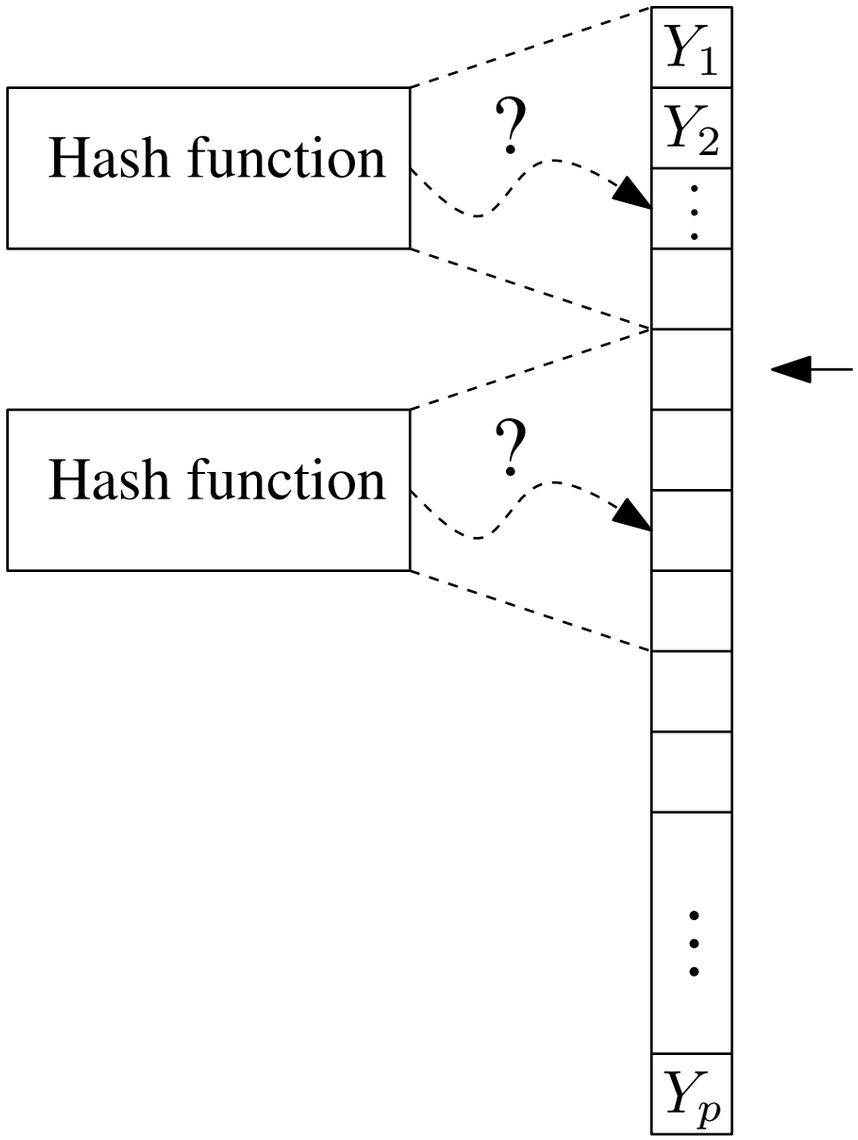




k-generators

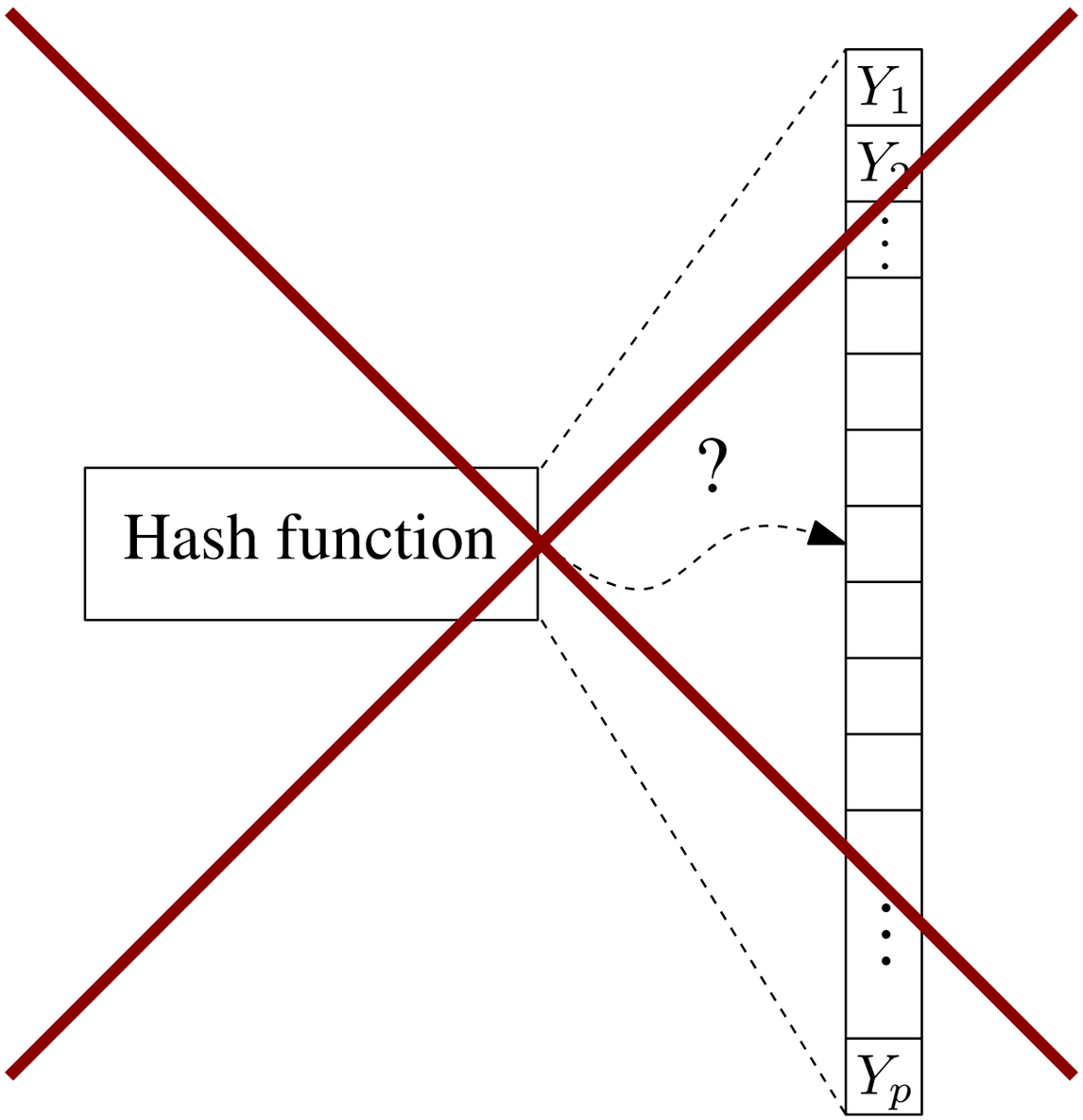
- Data structure for *sequential access* to a k-independent sequence
- Circumvent hashing lower bound
- Sequence of local hash functions evaluated over small domains

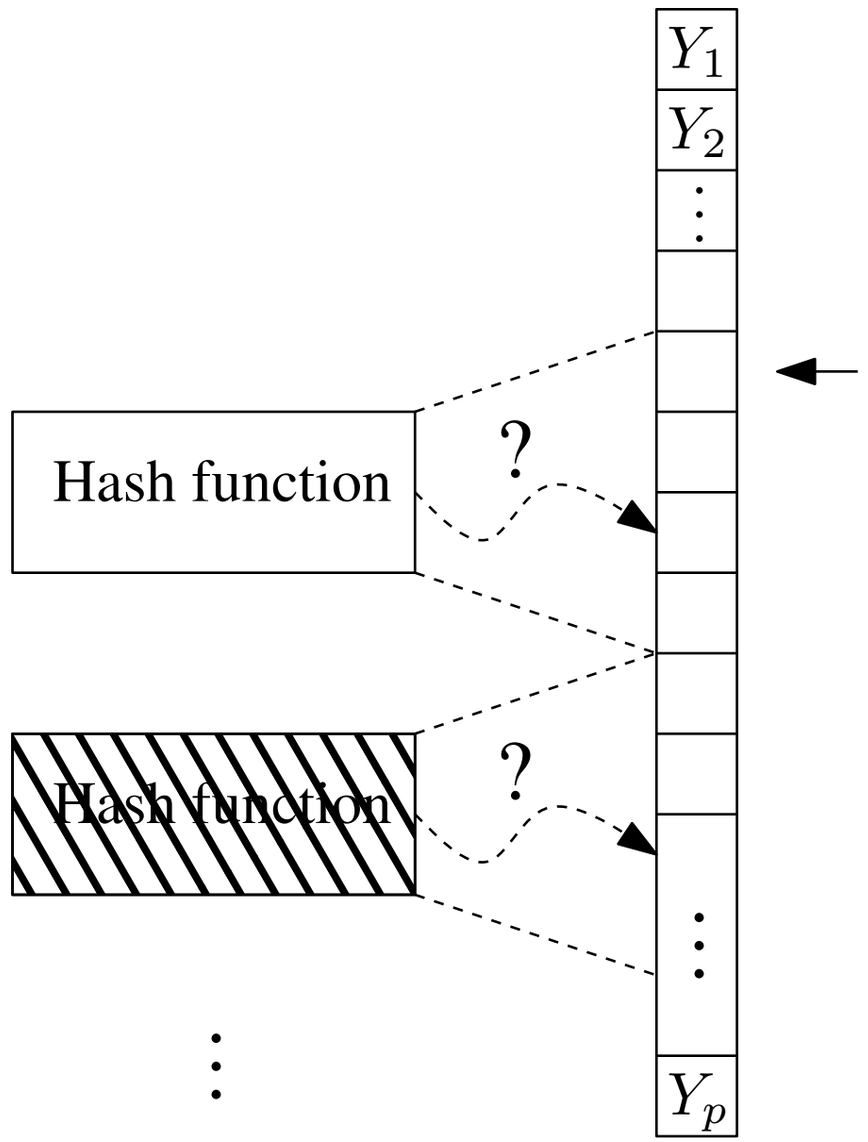




k-generators

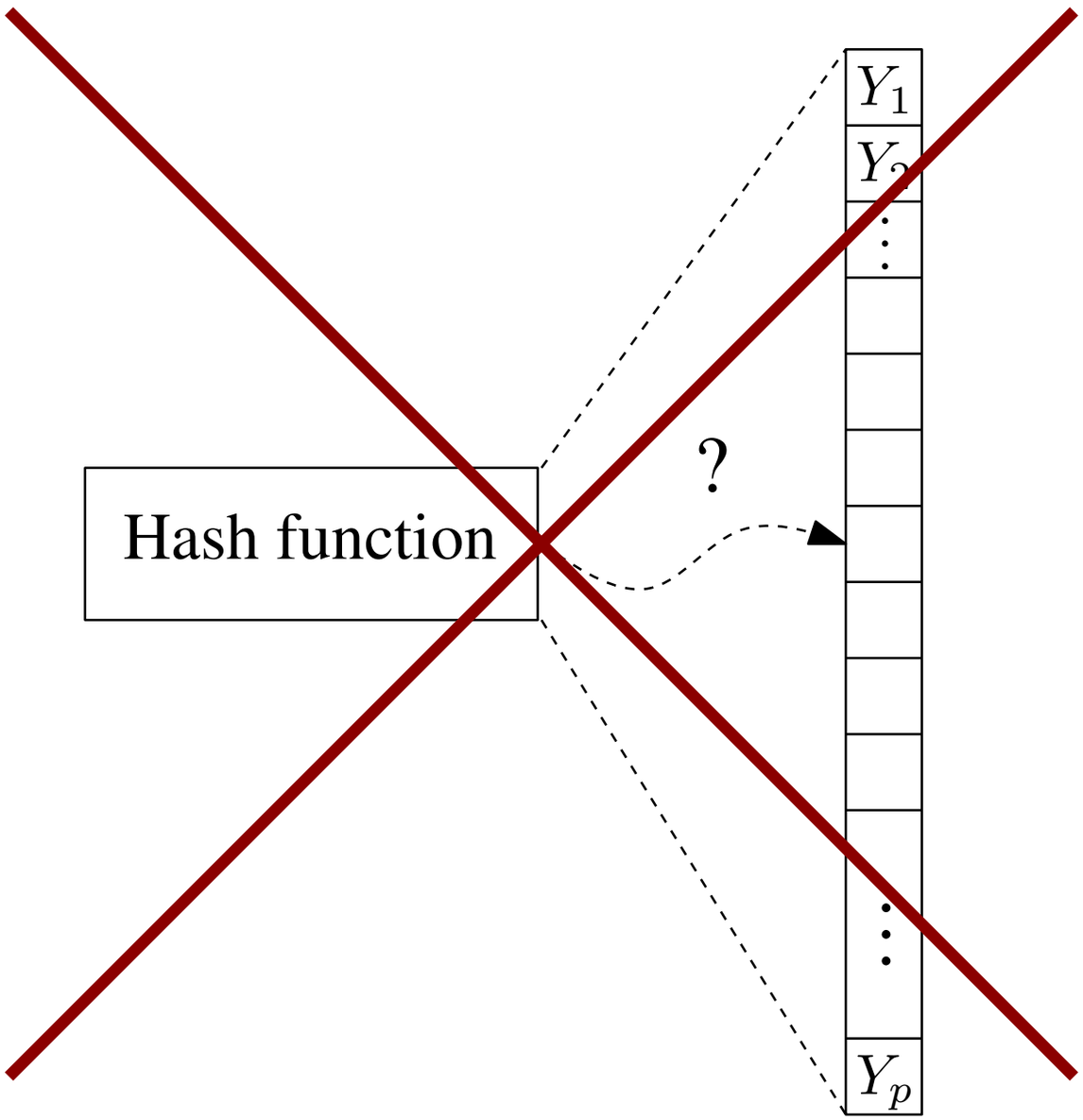
- Data structure for *sequential access* to a k-independent sequence
- Circumvent hashing lower bound
- Sequence of local hash functions evaluated over small domains

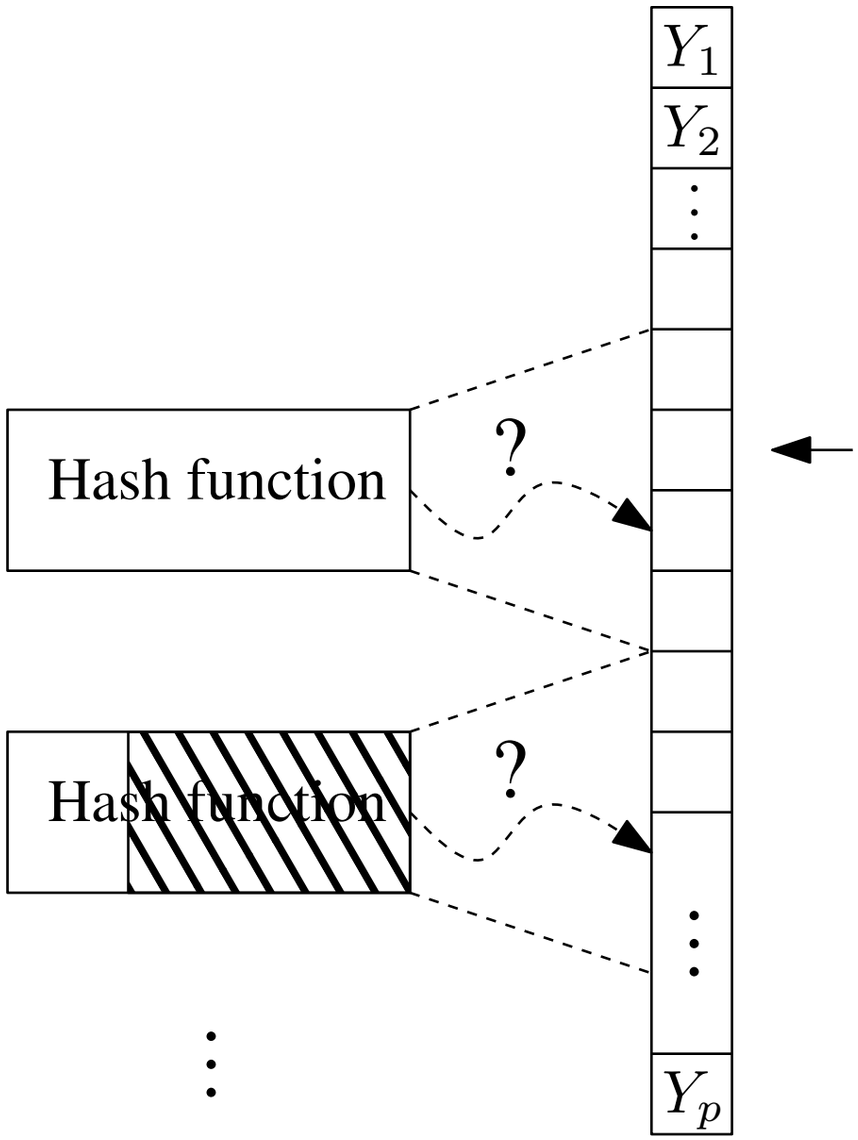




k-generators

- Data structure for *sequential access* to a k-independent sequence
- Circumvent hashing lower bound
- Sequence of local hash functions evaluated over small domains





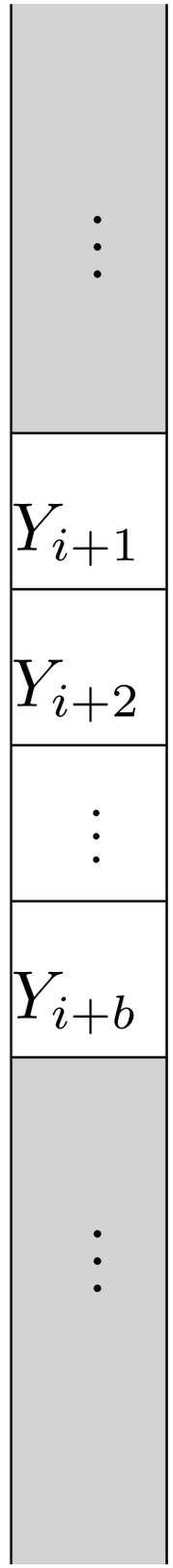
Construction overview

Initialization and evaluation of local hash functions:

Initialization and evaluation of local hash functions:

1. Multipoint evaluation

- k-independent polynomial hash function $h(x)$
  - Evaluate in k points using time $k \operatorname{poly} \log k$
  - $\operatorname{poly} \log k$ time per value generated

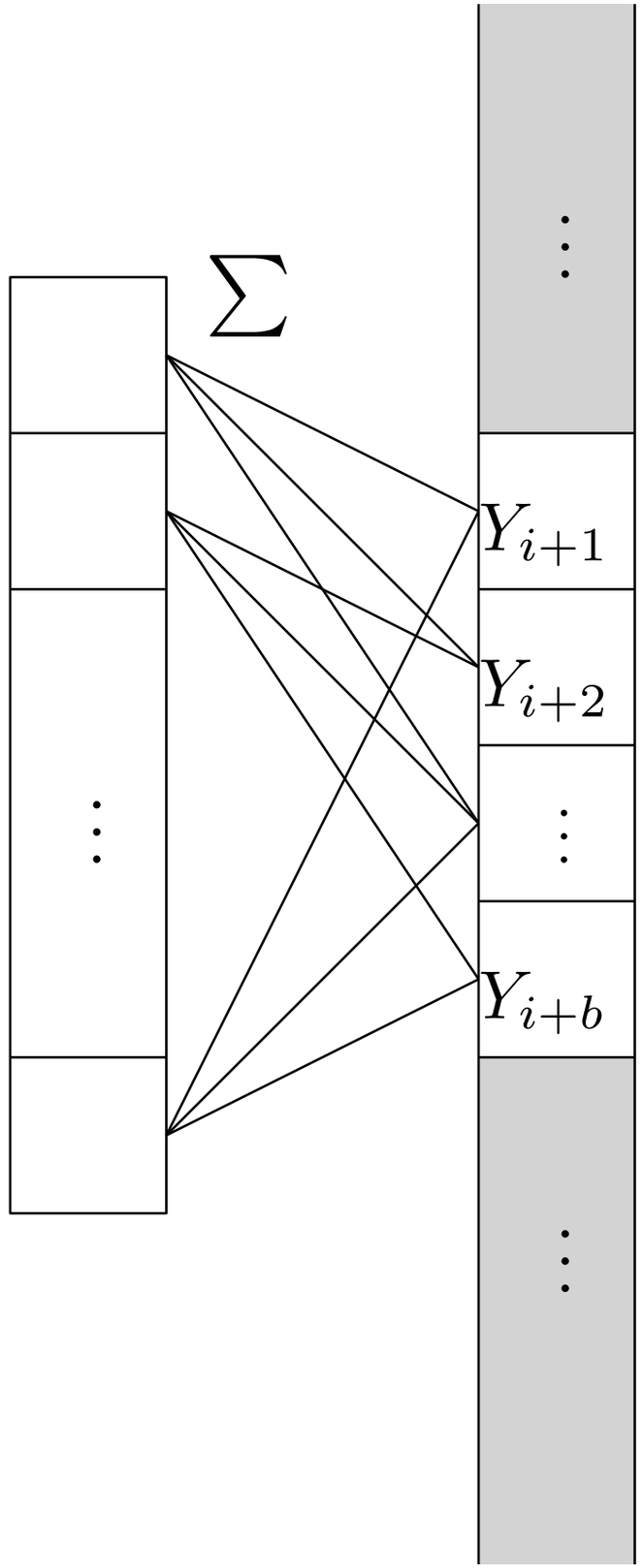


Construction overview

Initialization and evaluation of local hash functions:

1. Multipoint evaluation
   - k-independent polynomial hash function $h(x)$
     - Evaluate in k points using time $k \operatorname{poly} \log k$
     - $\operatorname{poly} \log k$ time per value generated

2. Expander hashing
   - Unbalanced bipartite expander, degree $O(1)$
     - Generate $k \operatorname{poly} \log k$ values from k values
     - Output is $\Omega(k)$-independent



Multipoint evaluation

- Assume that $\mathbb{F}_p$ contains a primitive kth root of unity ω_k

$$\omega_k : \begin{array}{l} \omega_k^k = 1 \\ \omega_k^i \neq 1 \text{ for } i = 1, \ldots, k-1 \end{array}$$

Multipoint evaluation

- Assume that $\mathbb{F}_p$ contains a primitive kth root of unity ω_k

$$\omega_k : \begin{array}{l} \omega_k^k = 1 \\ \omega_k^i \neq 1 \text{ for } i = 1, \ldots, k-1 \end{array}$$

$$h(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} \bmod p$$

- We can compute $h(1), h(\omega_k), \ldots, h(\omega_k^{k-1})$ in $O(k \log k)$ operations

Multipoint evaluation

- Assume that $\mathbb{F}_p$ contains a primitive kth root of unity ω_k

$$\omega_k : \begin{array}{l} \omega_k^k = 1 \\ \omega_k^i \neq 1 \text{ for } i = 1, \ldots, k-1 \end{array}$$

$$h(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} \bmod p$$

- We can compute $h(1), h(\omega_k), \ldots, h(\omega_k^{k-1})$ in $O(k \log k)$ operations

- Let ω be a primitive element that generates $\mathbb{F}_p^* = \mathbb{F}_p - \{0\}$

$$\mathbb{F}_p^* = \{1, \omega, \ldots, \omega_k, \omega\omega_k, \ldots, \omega_k^{k-1}, \omega\omega_k^{k-1}, \ldots, \omega^{p-2}\}$$

Multipoint evaluation

- Assume that $\mathbb{F}_p$ contains a primitive kth root of unity ω_k

$$\omega_k : \begin{array}{l} \omega_k^k = 1 \\ \omega_k^i \neq 1 \text{ for } i = 1, \ldots, k-1 \end{array}$$

$$h(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} \bmod p$$

- We can compute $h(1), h(\omega_k), \ldots, h(\omega_k^{k-1})$ in $O(k \log k)$ operations
- Let ω be a primitive element that generates $\mathbb{F}_p^* = \mathbb{F}_p - \{0\}$

$$\mathbb{F}_p^* = \{1, \omega, \ldots, \omega_k, \omega\omega_k, \ldots, \omega_k^{k-1}, \omega\omega_k^{k-1}, \ldots, \omega^{p-2}\}$$

$$h(1) \qquad h(\omega_k) \quad \cdots \quad h(\omega_k^{k-1})$$

Multipoint evaluation

- Assume that $\mathbb{F}_p$ contains a primitive kth root of unity ω_k

$$\omega_k : \begin{array}{l} \omega_k^k = 1 \\ \omega_k^i \neq 1 \text{ for } i = 1, \ldots, k-1 \end{array}$$

$$h(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} \bmod p$$

- We can compute $h(1), h(\omega_k), \ldots, h(\omega_k^{k-1})$ in $O(k \log k)$ operations
- Let ω be a primitive element that generates $\mathbb{F}_p^* = \mathbb{F}_p - \{0\}$

$$\mathbb{F}_p^* = \{1, \omega, \ldots, \omega_k, \omega\omega_k, \ldots, \omega_k^{k-1}, \omega\omega_k^{k-1}, \ldots, \omega^{p-2}\}$$

$$h(\omega) \qquad h(\omega\omega_k) \quad \cdots \quad h(\omega\omega_k^{k-1})$$

Multipoint evaluation

- Assume that $\mathbb{F}_p$ contains a primitive kth root of unity ω_k

$$\omega_k : \begin{array}{l} \omega_k^k = 1 \\ \omega_k^i \neq 1 \text{ for } i = 1, \ldots, k-1 \end{array}$$

$$h(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} \bmod p$$

- We can compute $h(1), h(\omega_k), \ldots, h(\omega_k^{k-1})$ in $O(k \log k)$ operations

- Let ω be a primitive element that generates $\mathbb{F}_p^* = \mathbb{F}_p - \{0\}$

$$\mathbb{F}_p^* = \{1, \omega, \ldots, \omega_k, \omega\omega_k, \ldots, \omega_k^{k-1}, \omega\omega_k^{k-1}, \ldots, \omega^{p-2}\}$$

$$h(\omega) \qquad h(\omega\omega_k) \quad \cdots \quad h(\omega\omega_k^{k-1})$$

- Construct $h'(x) = \sum_{i=0}^{k-1} a_i \omega^i x^i$ compute $h'(1), h'(\omega_k), \ldots, h'(\omega_k^{k-1})$

Multipoint evaluation

- Assume that $\mathbb{F}_p$ contains a primitive kth root of unity ω_k

$$\omega_k : \begin{array}{l} \omega_k^k = 1 \\ \omega_k^i \neq 1 \text{ for } i = 1, \ldots, k-1 \end{array}$$

$$h(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} \bmod p$$

- We can compute $h(1), h(\omega_k), \ldots, h(\omega_k^{k-1})$ in $O(k \log k)$ operations

- Let ω be a primitive element that generates $\mathbb{F}_p^* = \mathbb{F}_p - \{0\}$

$$\mathbb{F}_p^* = \{1, \omega, \ldots, \omega_k, \omega\omega_k, \ldots, \omega_k^{k-1}, \omega\omega_k^{k-1}, \ldots, \omega^{p-2}\}$$

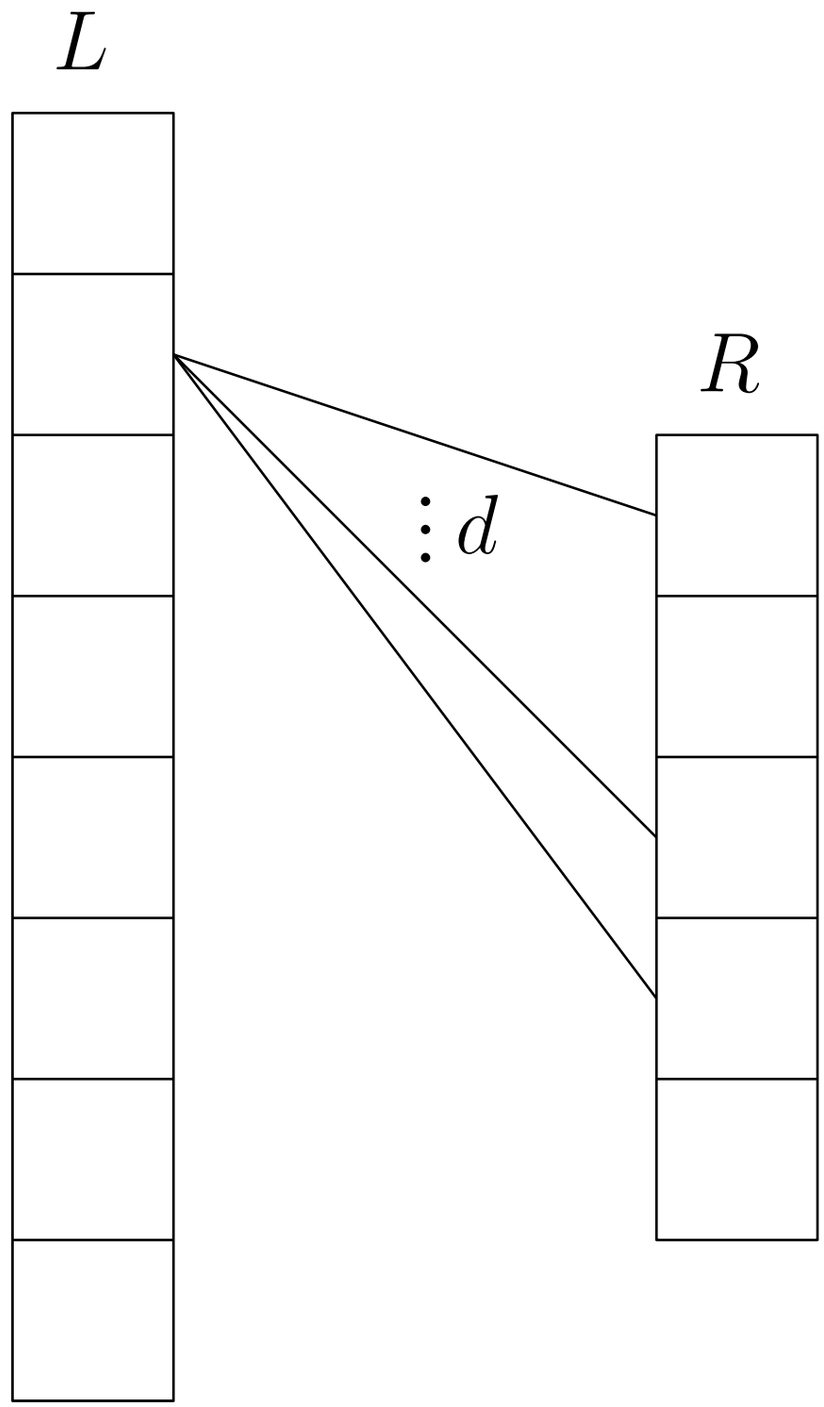
$$h(\omega) \qquad h(\omega\omega_k) \quad \cdots \quad h(\omega\omega_k^{k-1})$$

- Construct $h'(x) = \sum_{i=0}^{k-1} a_i \omega^i x^i$ compute $h'(1), h'(\omega_k), \ldots, h'(\omega_k^{k-1})$

- General multipoint evaluation: Time $O(k \log^3 k)$ [GG'13]

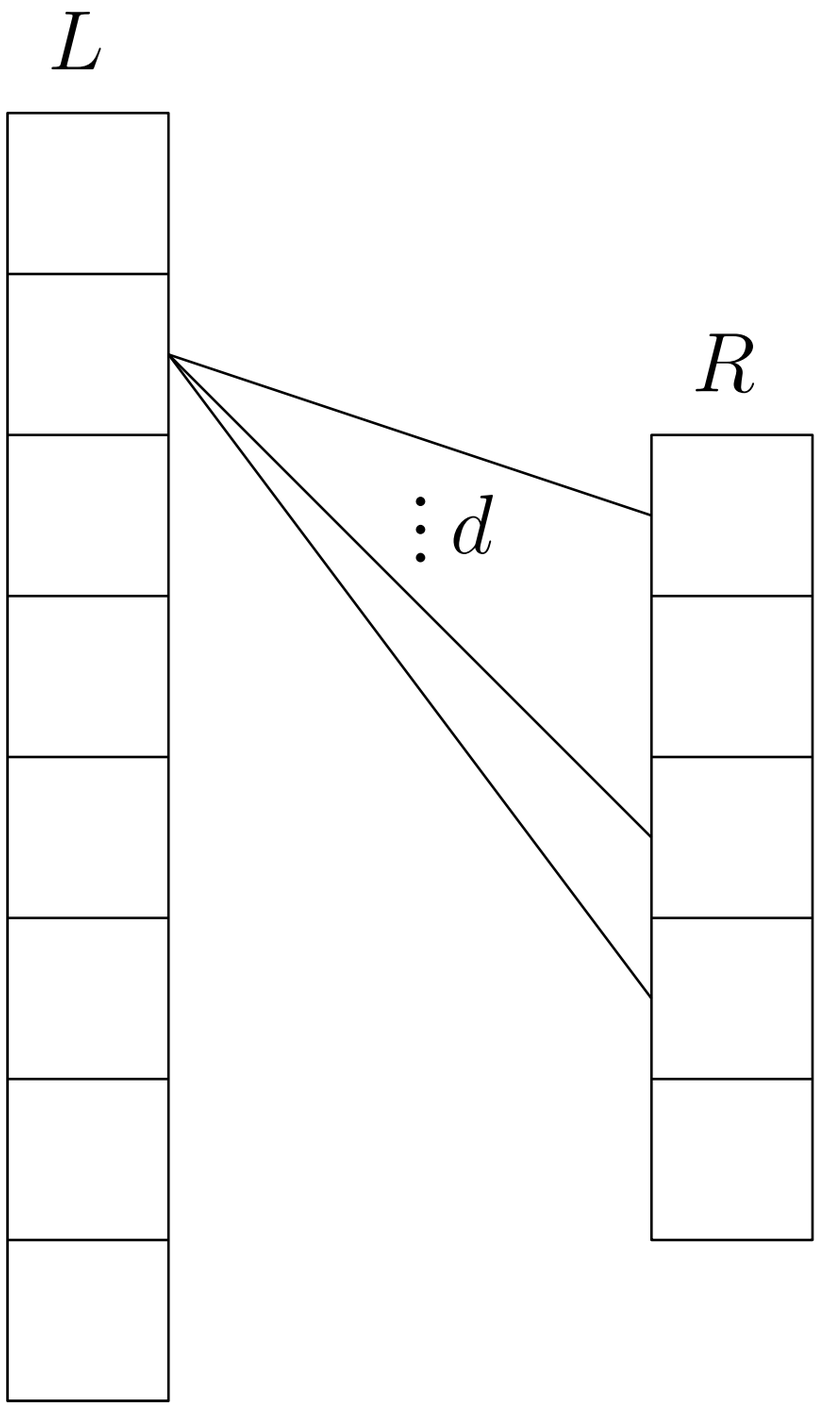
Expander hashing

- Left d-regular bipartite graph Γ



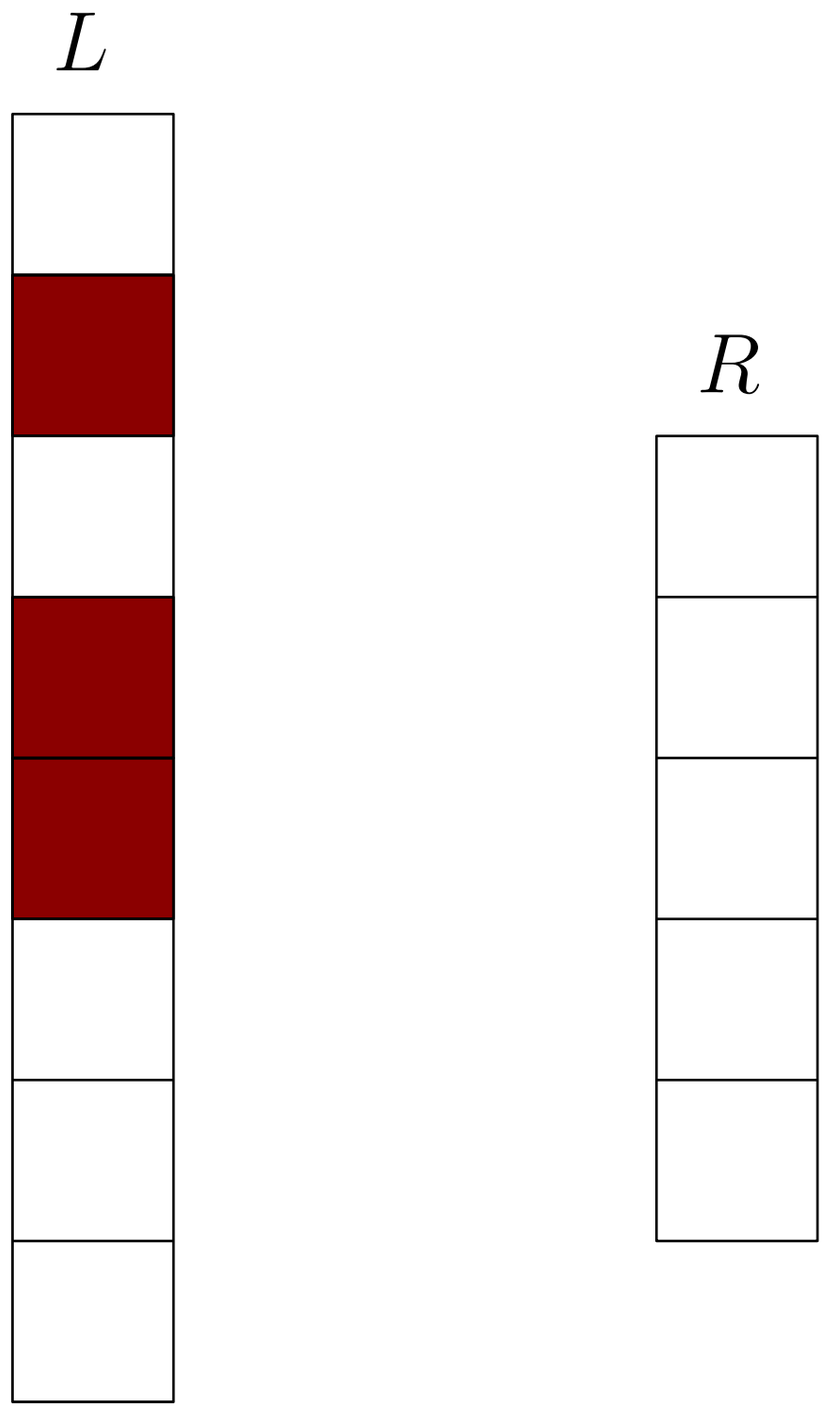
Expander hashing

- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor



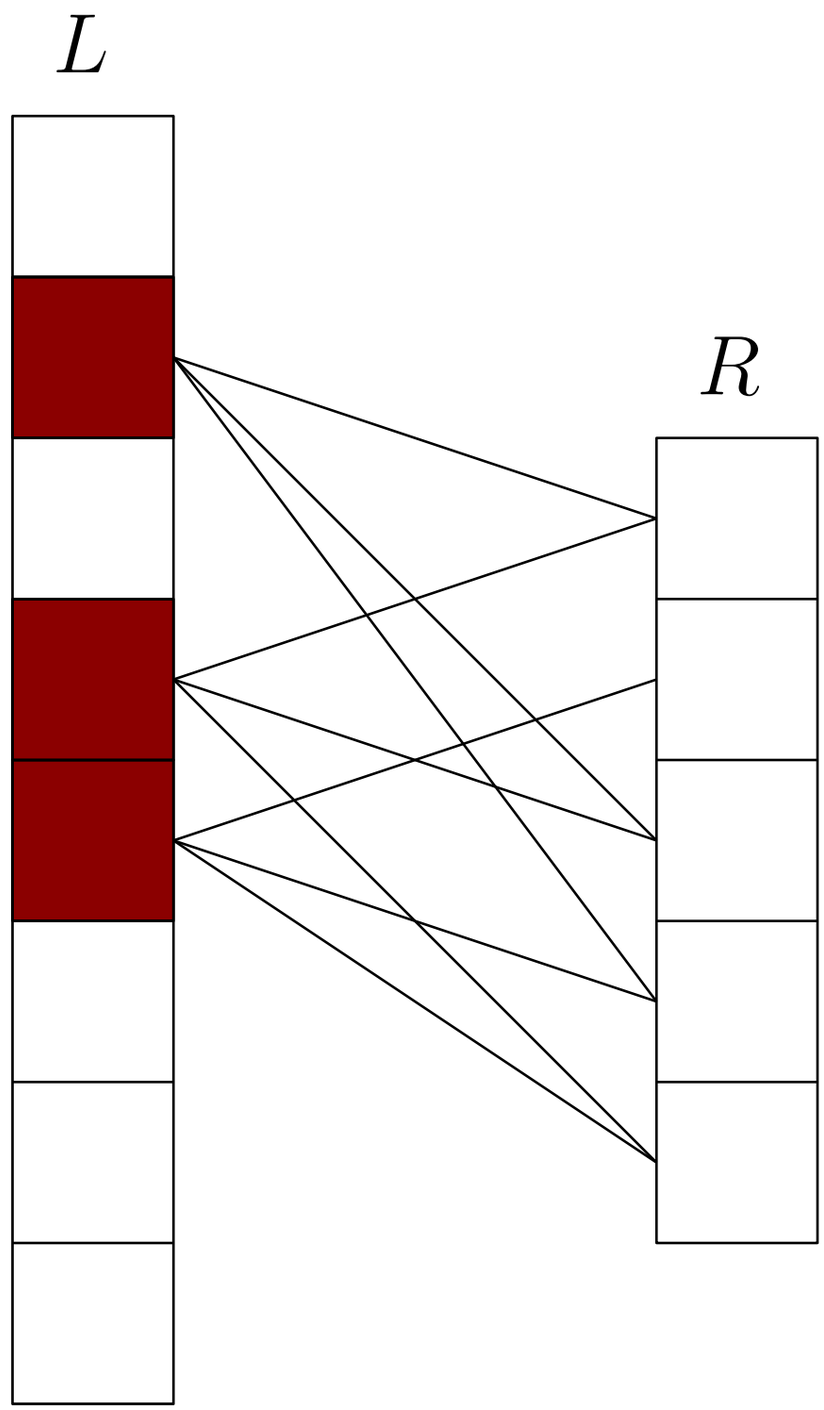
Expander hashing

- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor
- Example for $|S| = 3$



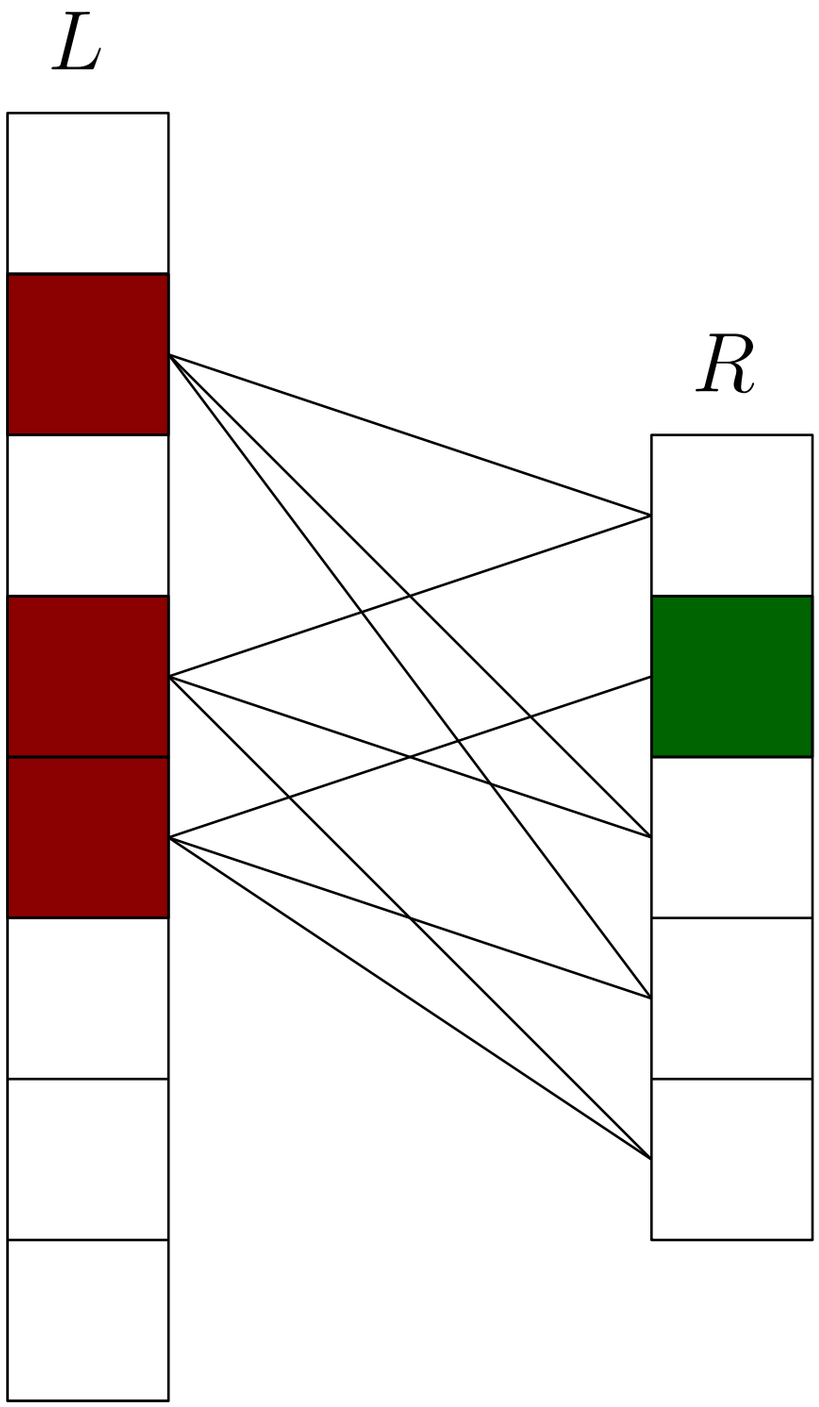
Expander hashing

- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor
- Example for $|S| = 3$



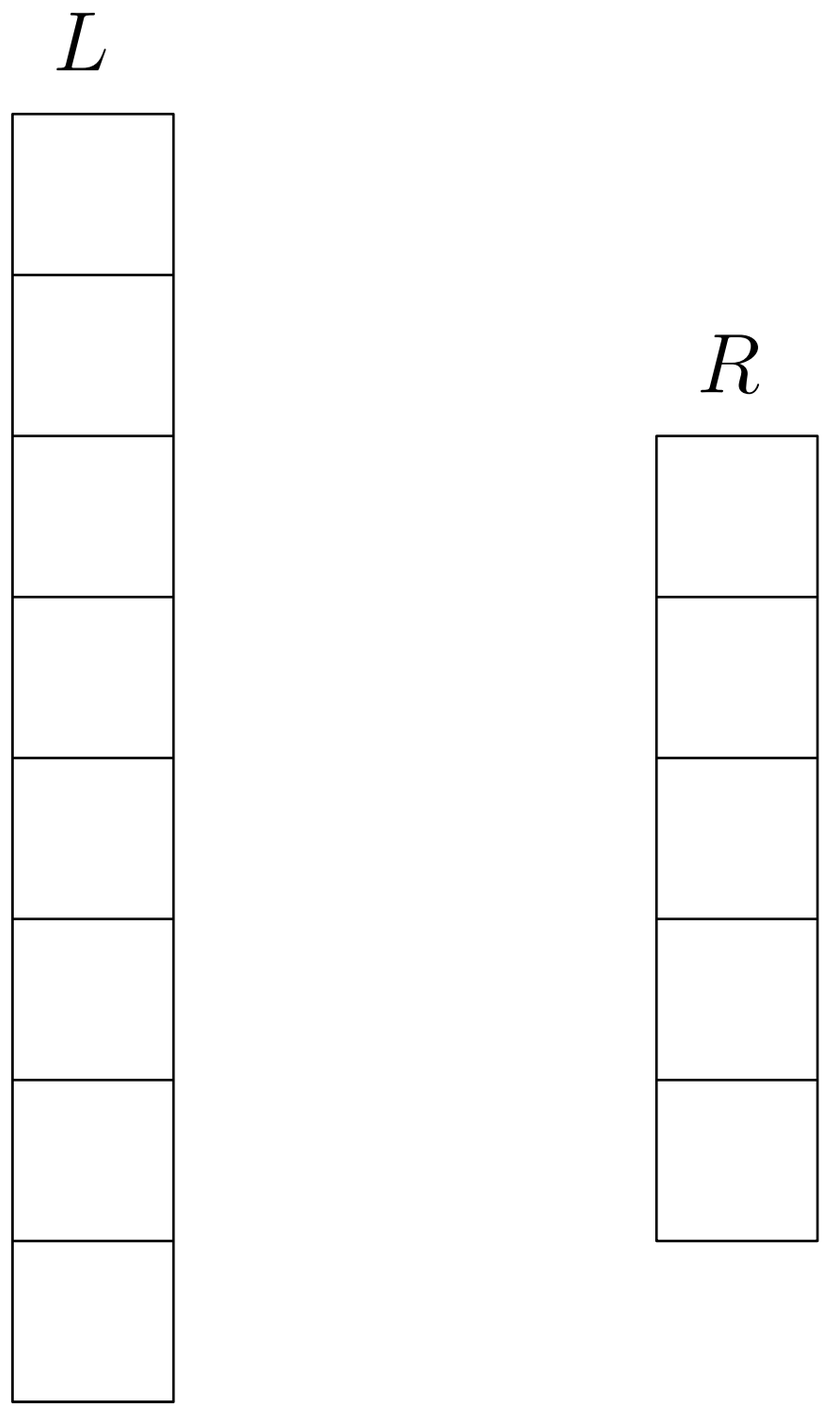
Expander hashing

- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor
- Example for $|S| = 3$



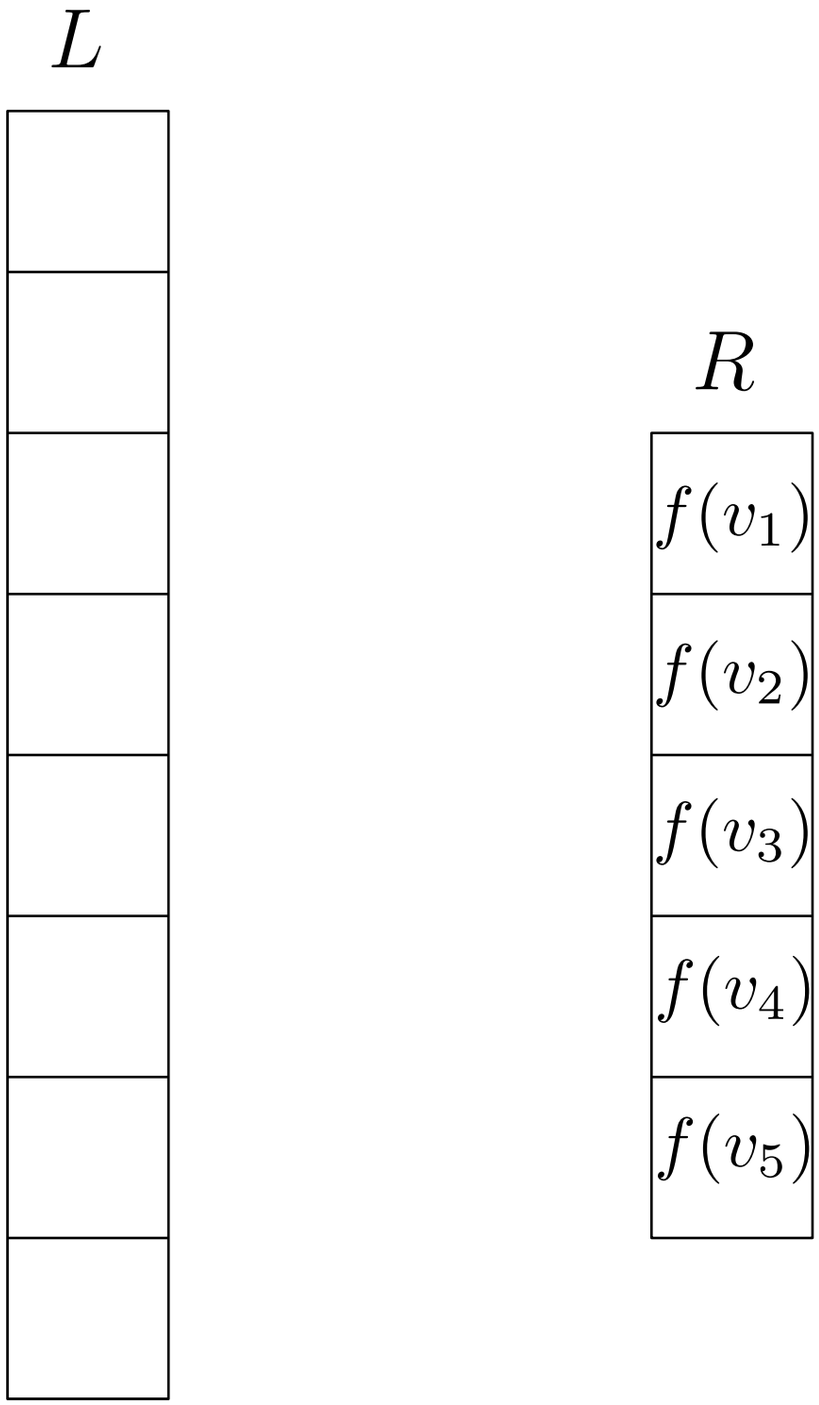
Expander hashing

- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor
- Example for $|S| = 3$
- Expander hashing [Siegel'89]



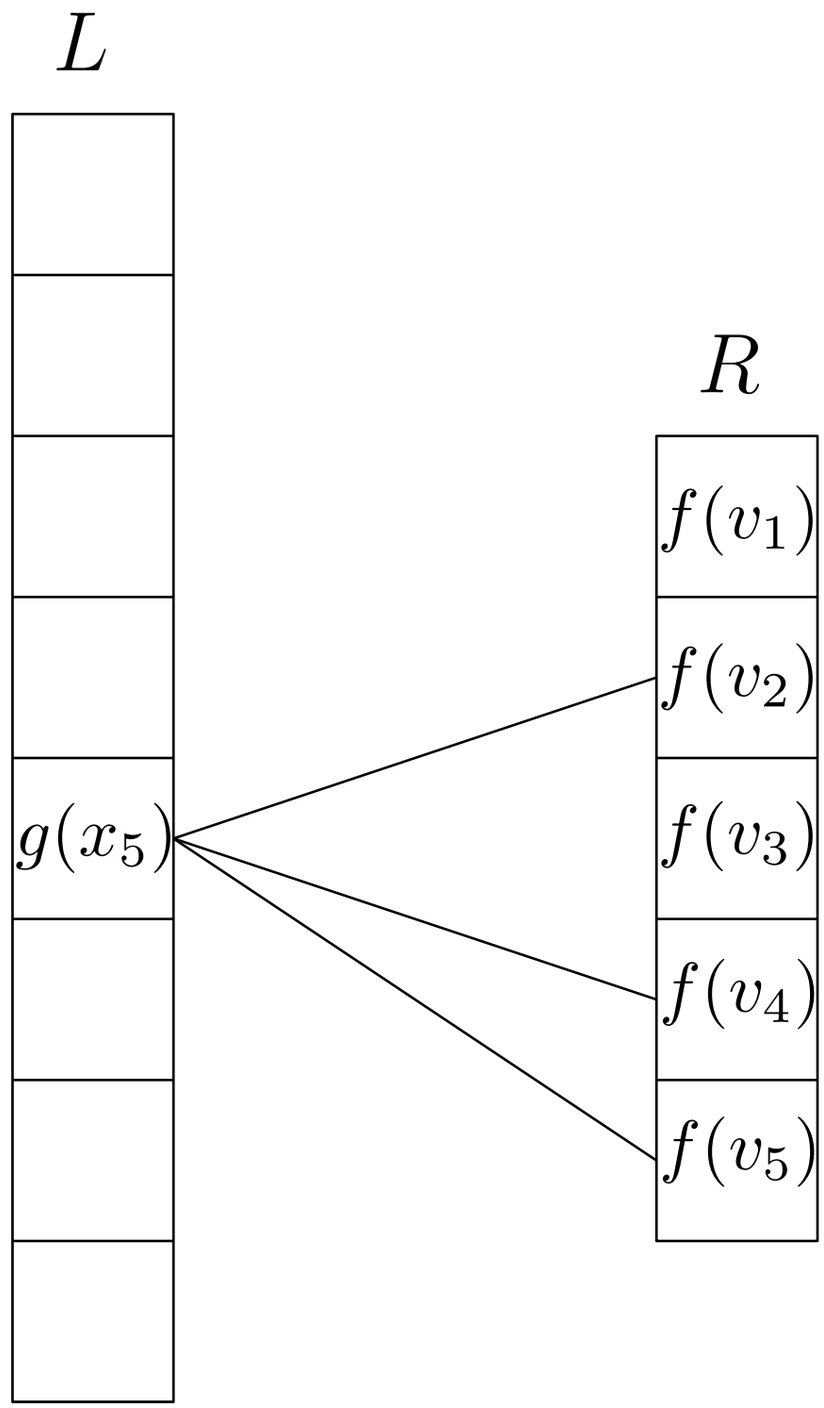
Expander hashing

- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor
- Example for $|S| = 3$
- Expander hashing [Siegel'89]
  - Let $f : R \to \mathbb{F}_p$ be dk-independent



Expander hashing

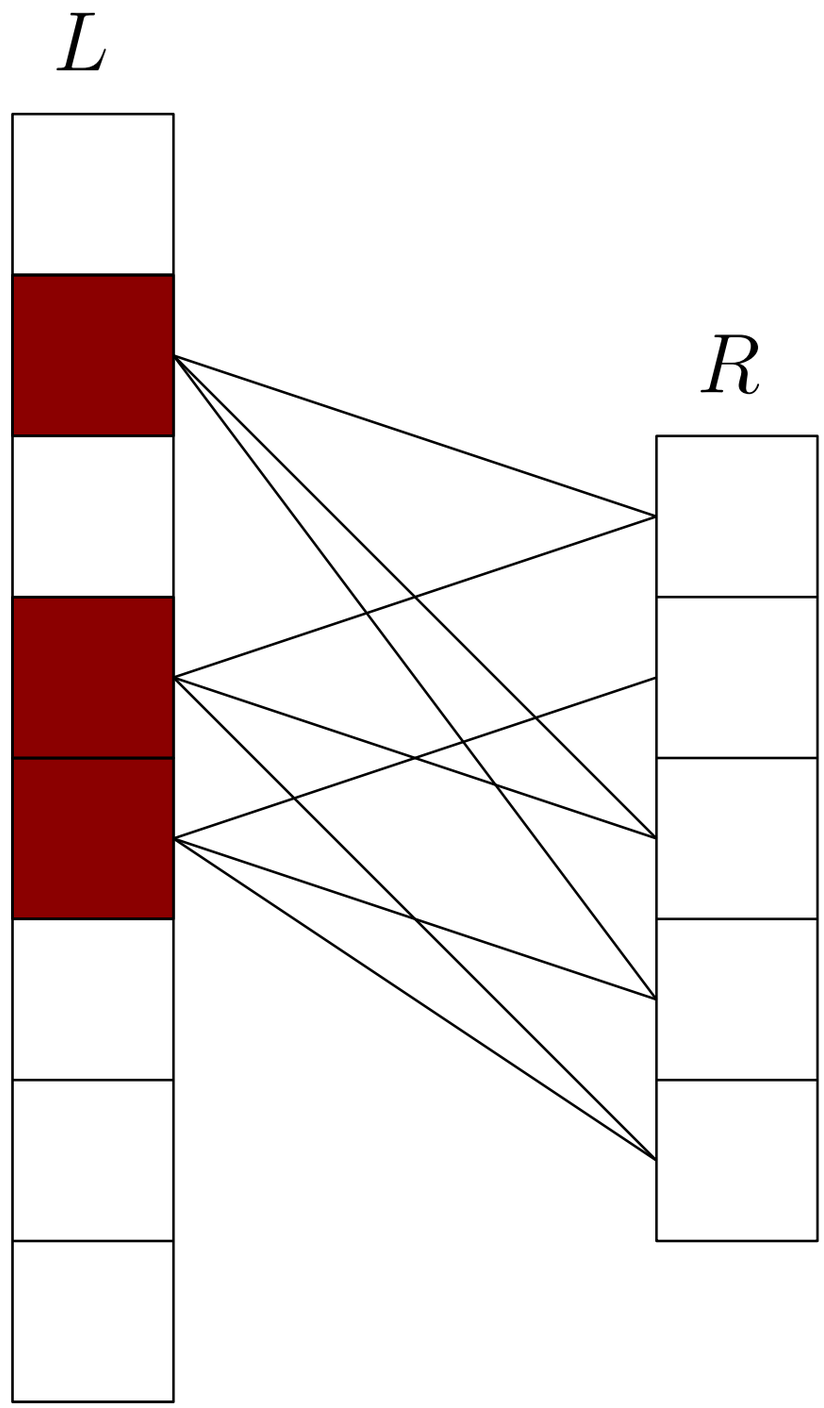
- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor
- Example for $|S| = 3$
- Expander hashing [Siegel'89]
  - Let $f : R \to \mathbb{F}_p$ be dk-independent
  - $g(x) = \sum_{v \in \Gamma(\{x\})} f(v)$ is k-independent



$$g(x_5) = f(v_2) + f(v_4) + f(v_5) \bmod p$$

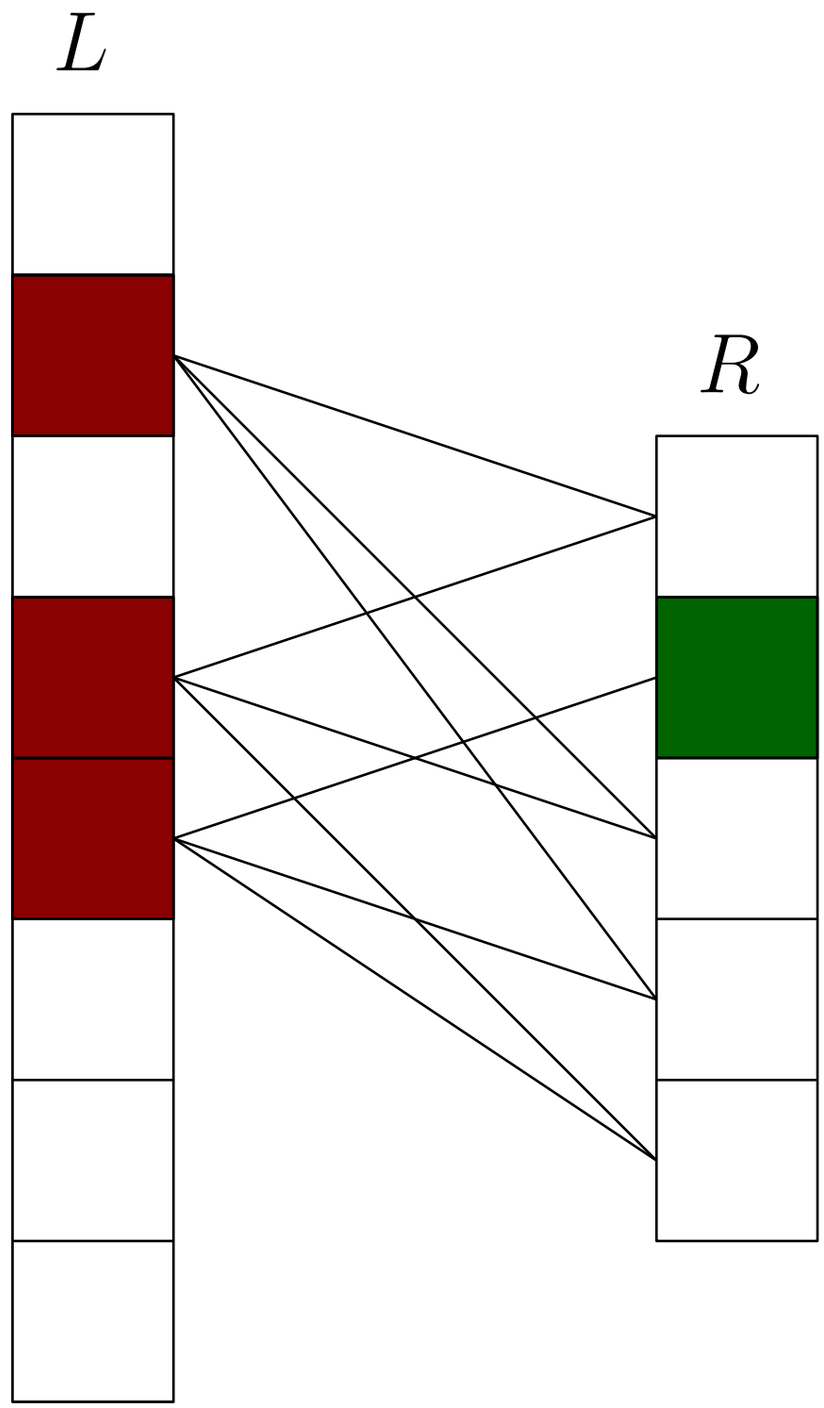
Expander hashing

- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor
- Example for $|S| = 3$
- Expander hashing [Siegel'89]
  - Let $f : R \to \mathbb{F}_p$ be dk-independent
  - $g(x) = \sum_{v \in \Gamma(\{x\})} f(v)$ is k-independent



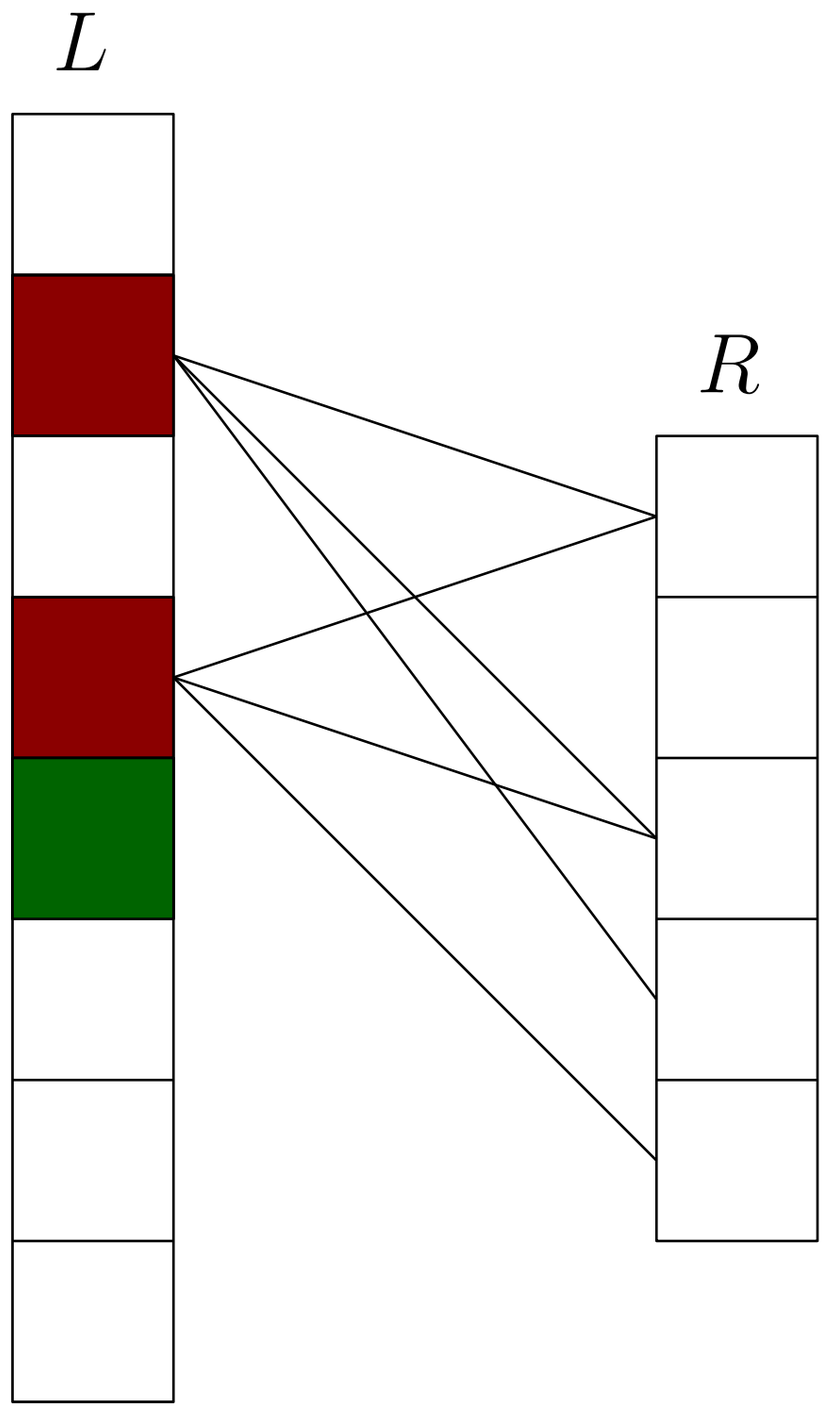
Expander hashing

- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor
- Example for $|S| = 3$
- Expander hashing [Siegel'89]
  - Let $f : R \to \mathbb{F}_p$ be dk-independent
  - $g(x) = \sum_{v \in \Gamma(\{x\})} f(v)$ is k-independent



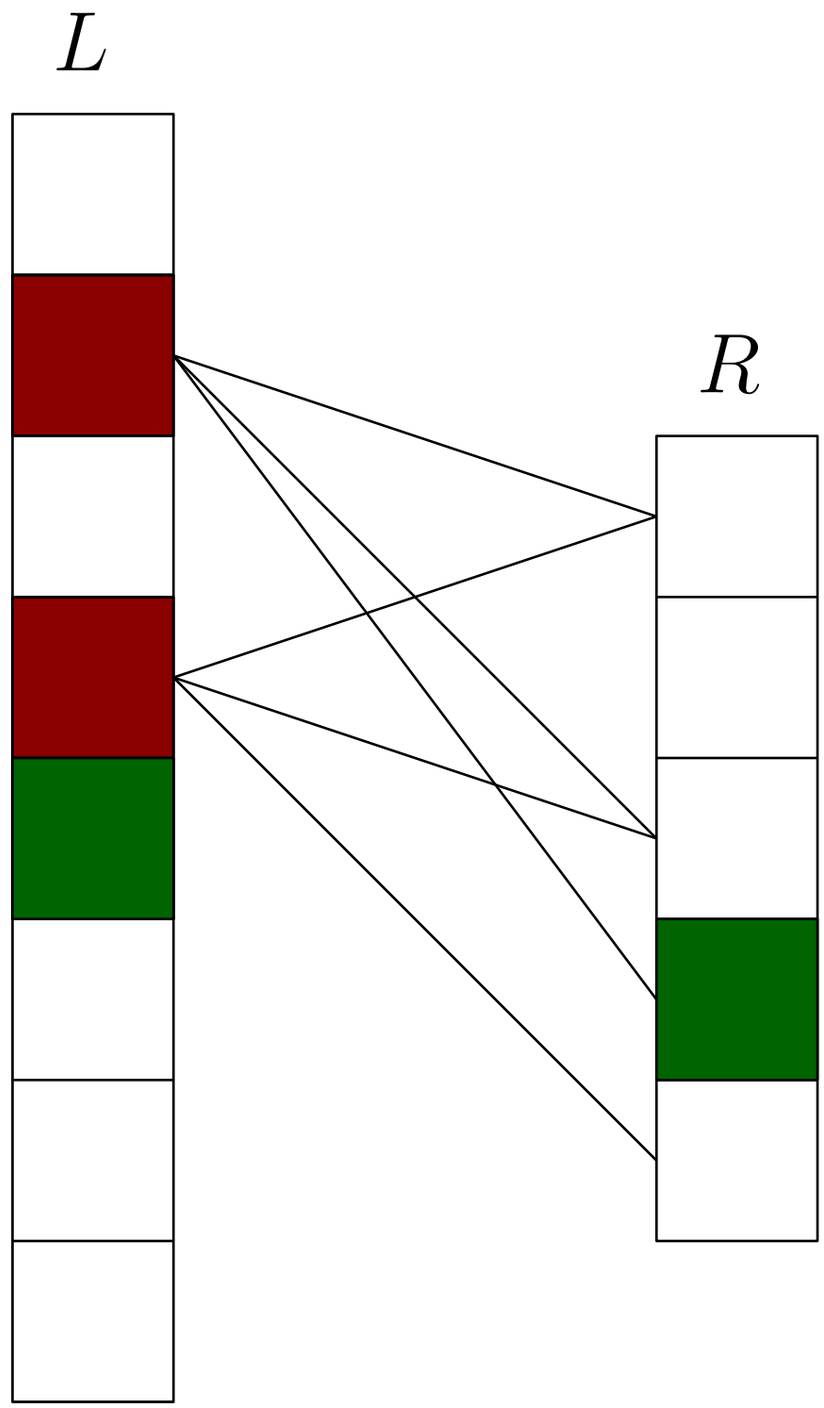
Expander hashing

- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor
- Example for $|S| = 3$
- Expander hashing [Siegel'89]
  - Let $f : R \to \mathbb{F}_p$ be dk-independent
  - $g(x) = \sum_{v \in \Gamma(\{x\})} f(v)$ is k-independent



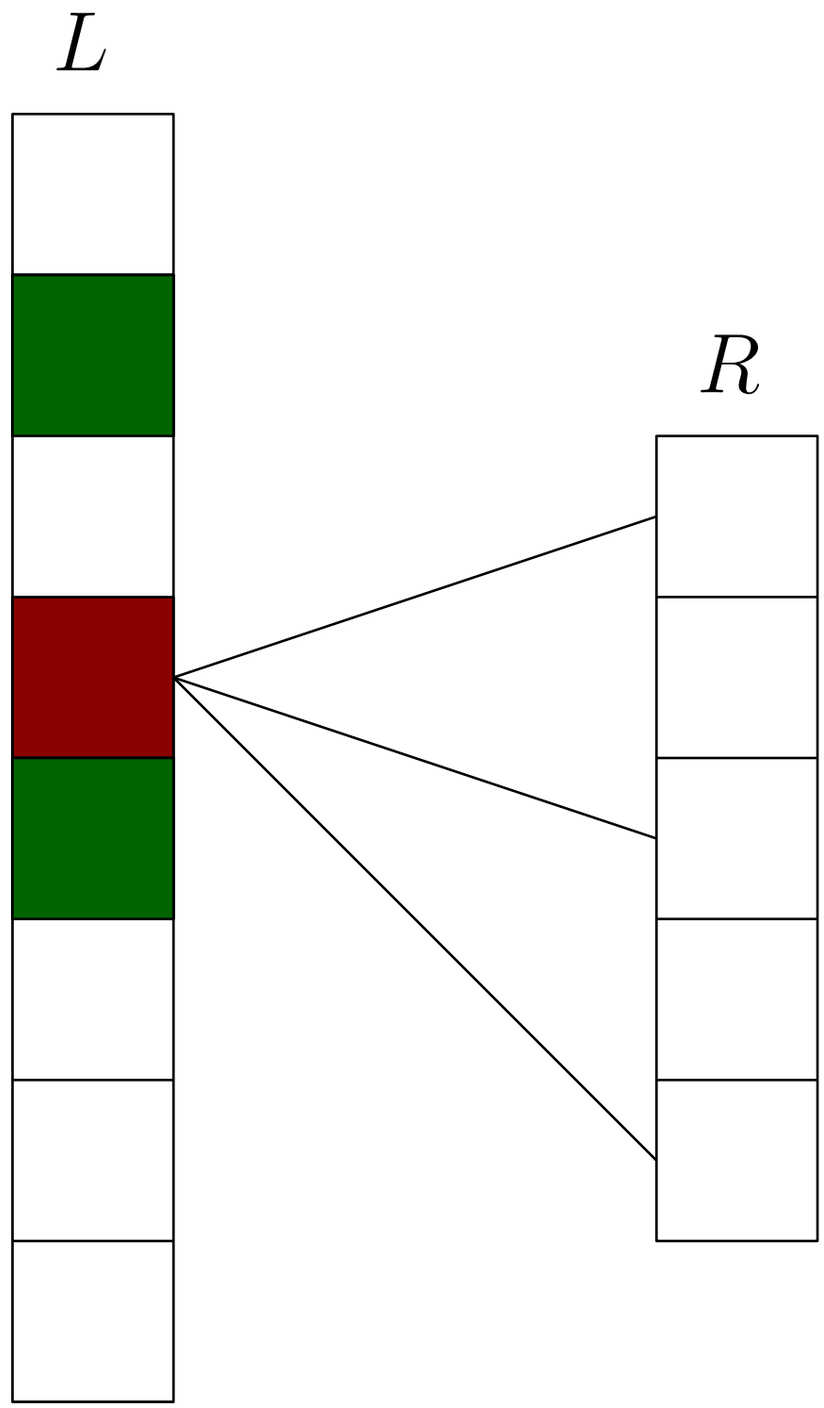
Expander hashing

- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor
- Example for $|S| = 3$
- Expander hashing [Siegel'89]
  - Let $f : R \to \mathbb{F}_p$ be dk-independent
  - $g(x) = \sum_{v \in \Gamma(\{x\})} f(v)$ is k-independent



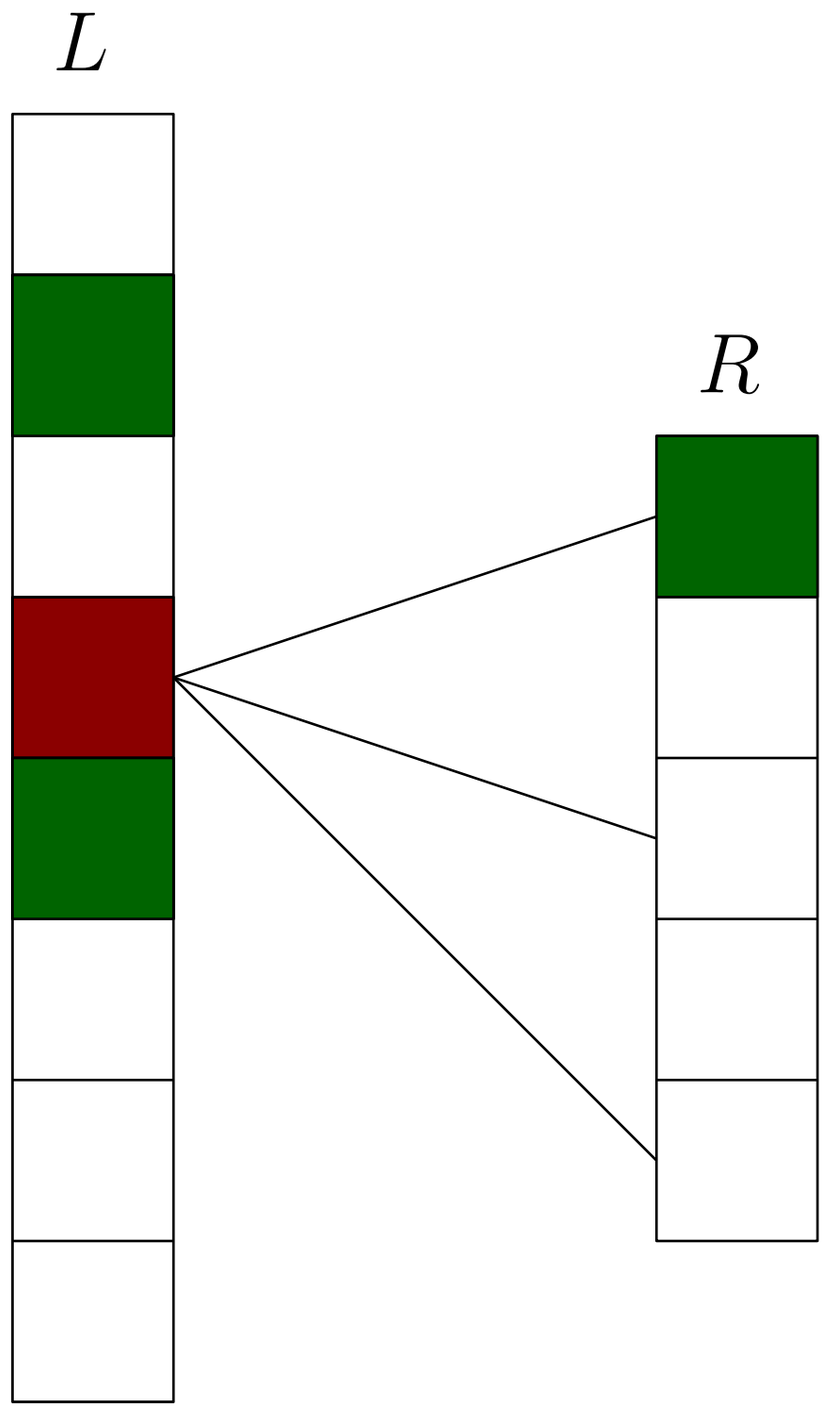
Expander hashing

- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor
- Example for $|S| = 3$
- Expander hashing [Siegel'89]
  - Let $f : R \to \mathbb{F}_p$ be dk-independent
  - $g(x) = \sum_{v \in \Gamma(\{x\})} f(v)$ is k-independent



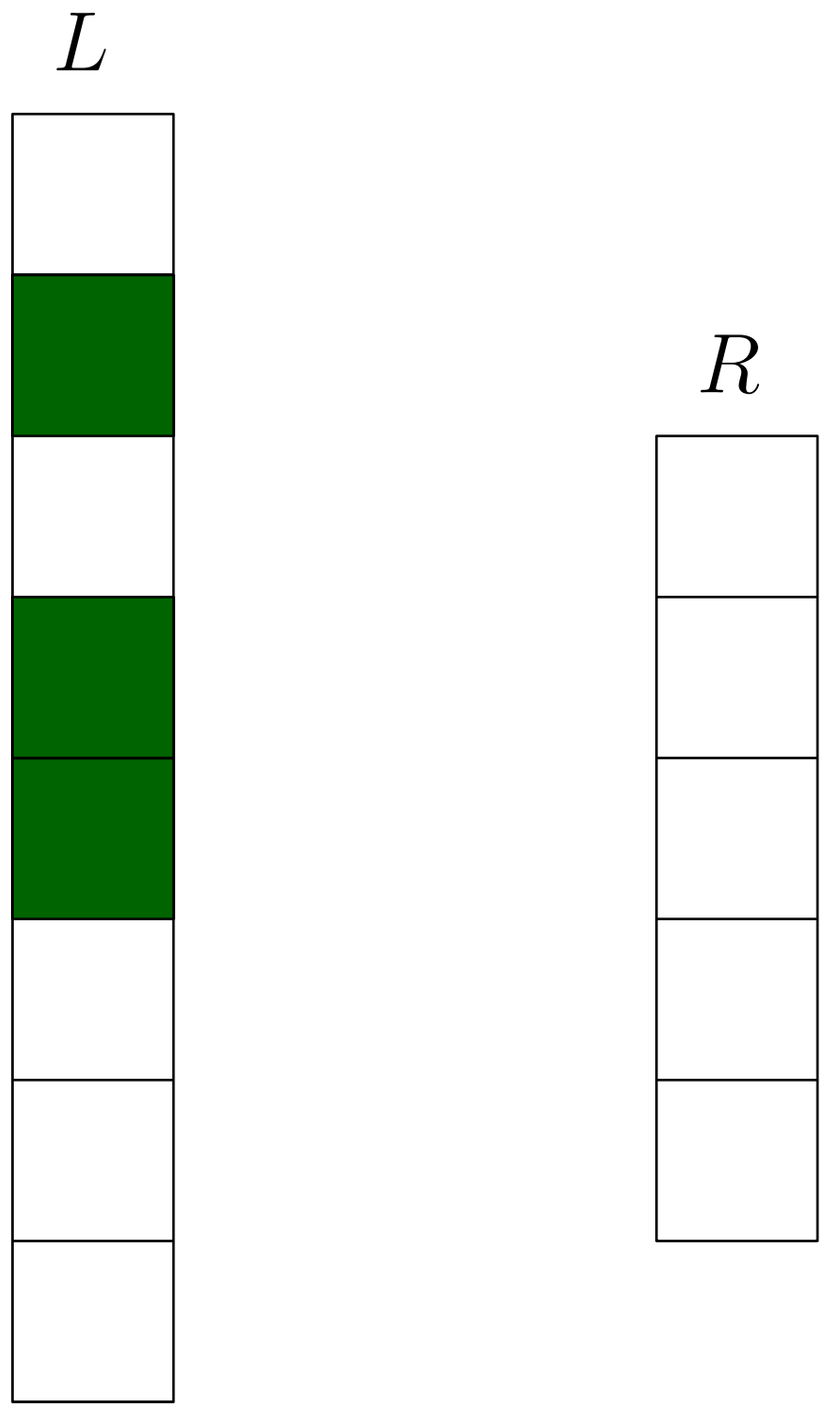
Expander hashing

- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor
- Example for $|S| = 3$
- Expander hashing [Siegel'89]
  - Let $f : R \to \mathbb{F}_p$ be dk-independent
  - $g(x) = \sum_{v \in \Gamma(\{x\})} f(v)$ is k-independent



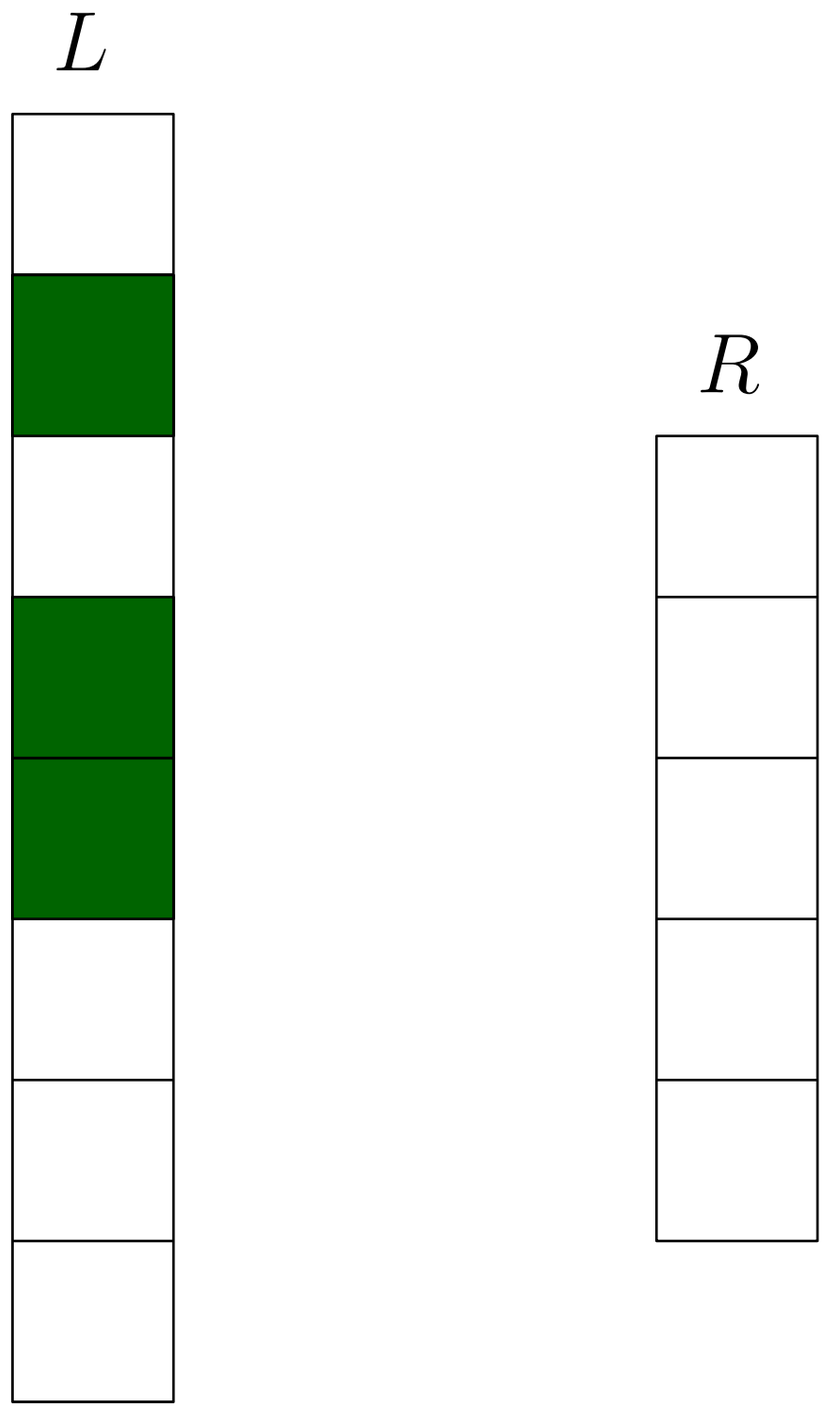
Expander hashing

- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor
- Example for $|S| = 3$
- Expander hashing [Siegel'89]
  - Let $f : R \to \mathbb{F}_p$ be dk-independent
  - $g(x) = \sum_{v \in \Gamma(\{x\})} f(v)$ is k-independent



Expander hashing

- Left d-regular bipartite graph Γ
- Γ is k-unique if every $S \subseteq L$ with $1 \leq |S| \leq k$ has a unique neighbor
- Example for $|S| = 3$
- Expander hashing [Siegel'89]
  - Let $f : R \to \mathbb{F}_p$ be dk-independent
  - $g(x) = \sum_{v \in \Gamma(\{x\})} f(v)$ is k-independent



Claim:

- There exists k-unique Γ with
  - $|R| = O(k \log k)$
  - $|L| = (\text{poly} \log k) \cdot |R|$
  - $d = O(1)$

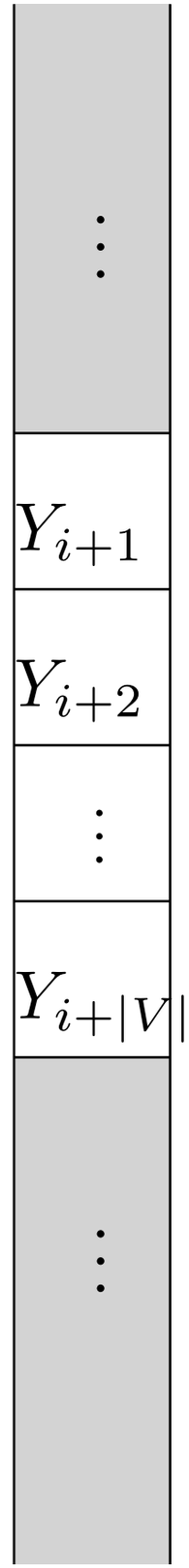
Example construction

Initialization and evaluation of local hash functions:

Initialization and evaluation of local hash functions:

1. Multipoint evaluation

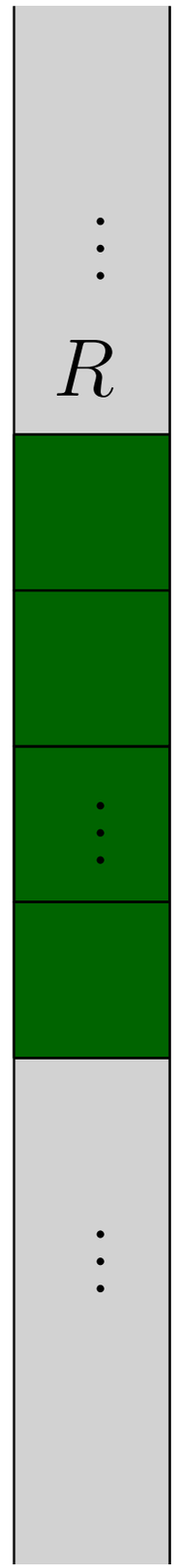
- dk-independent polynomial hash function $h(x)$
  - Generate $|R| = \Omega(k \log k)$ values using time $O(k \log^4 k)$



Example construction

Initialization and evaluation of local hash functions:

1. Multipoint evaluation
   - dk-independent polynomial hash function $h(x)$
     - Generate $|R| = \Omega(k \log k)$ values using time $O(k \log^4 k)$

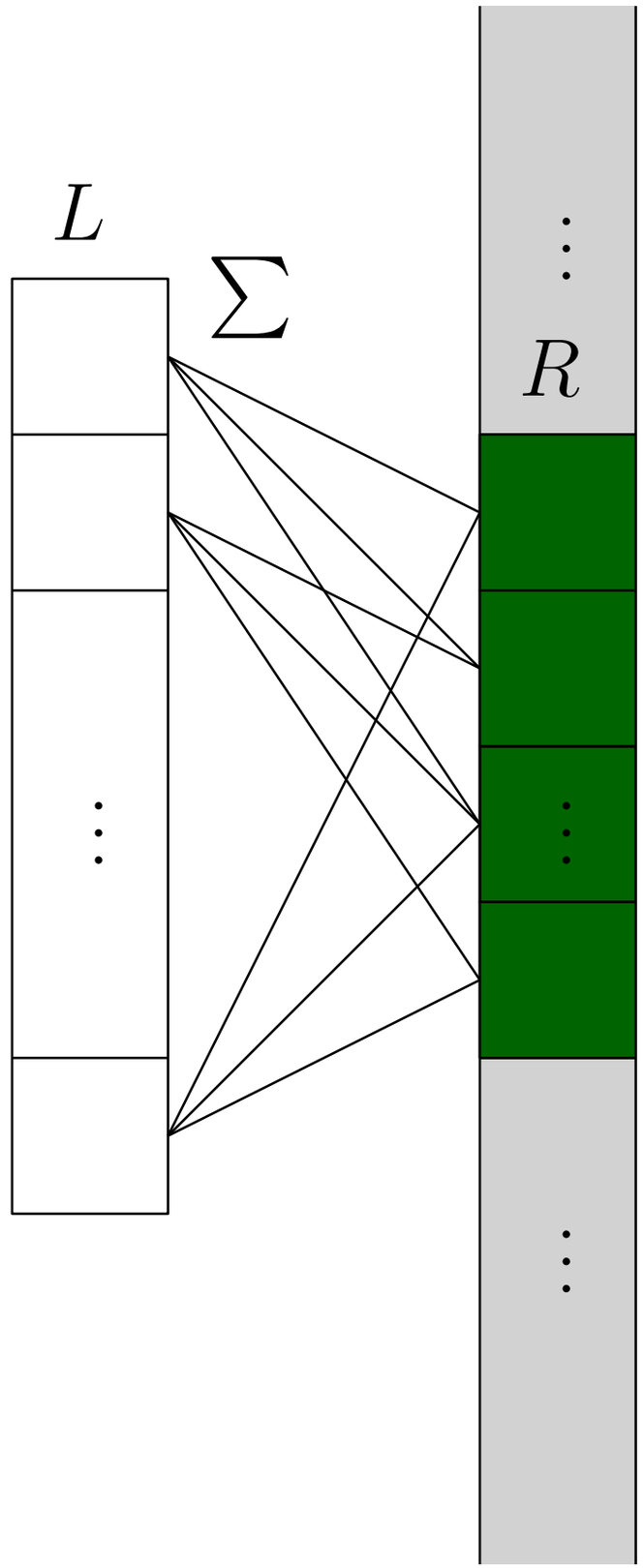


Example construction

Initialization and evaluation of local hash functions:

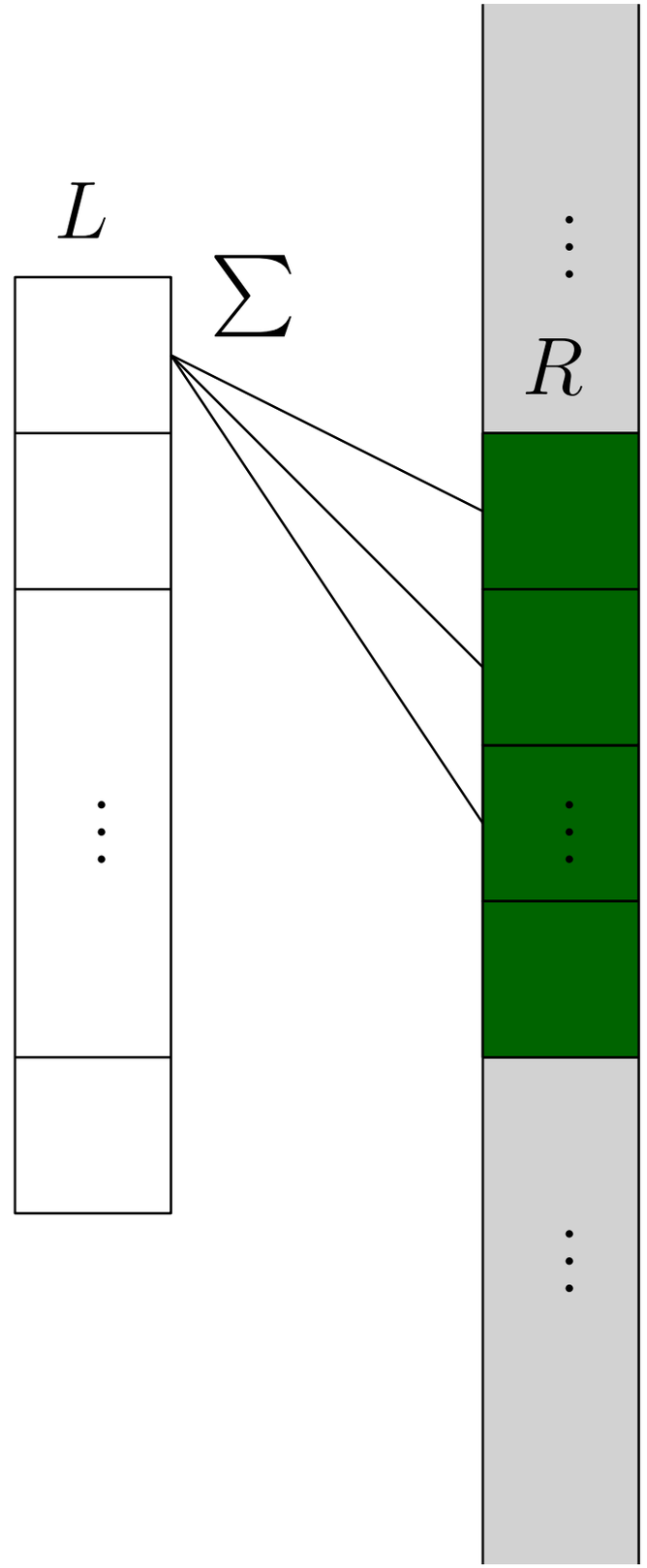
1. Multipoint evaluation
   - dk-independent polynomial hash function $h(x)$
     - Generate $|R| = \Omega(k \log k)$ values using time $O(k \log^4 k)$

2. Expander hashing
   - Construct random Γ with:
     - Constant degree $d = 8$
     - $|L| = (\log^4 k)|R|$



Example construction

Initialization and evaluation of local hash functions:



1. Multipoint evaluation
   - dk-independent polynomial hash function $h(x)$
     - Generate $|R| = \Omega(k \log k)$ values using time $O(k \log^4 k)$

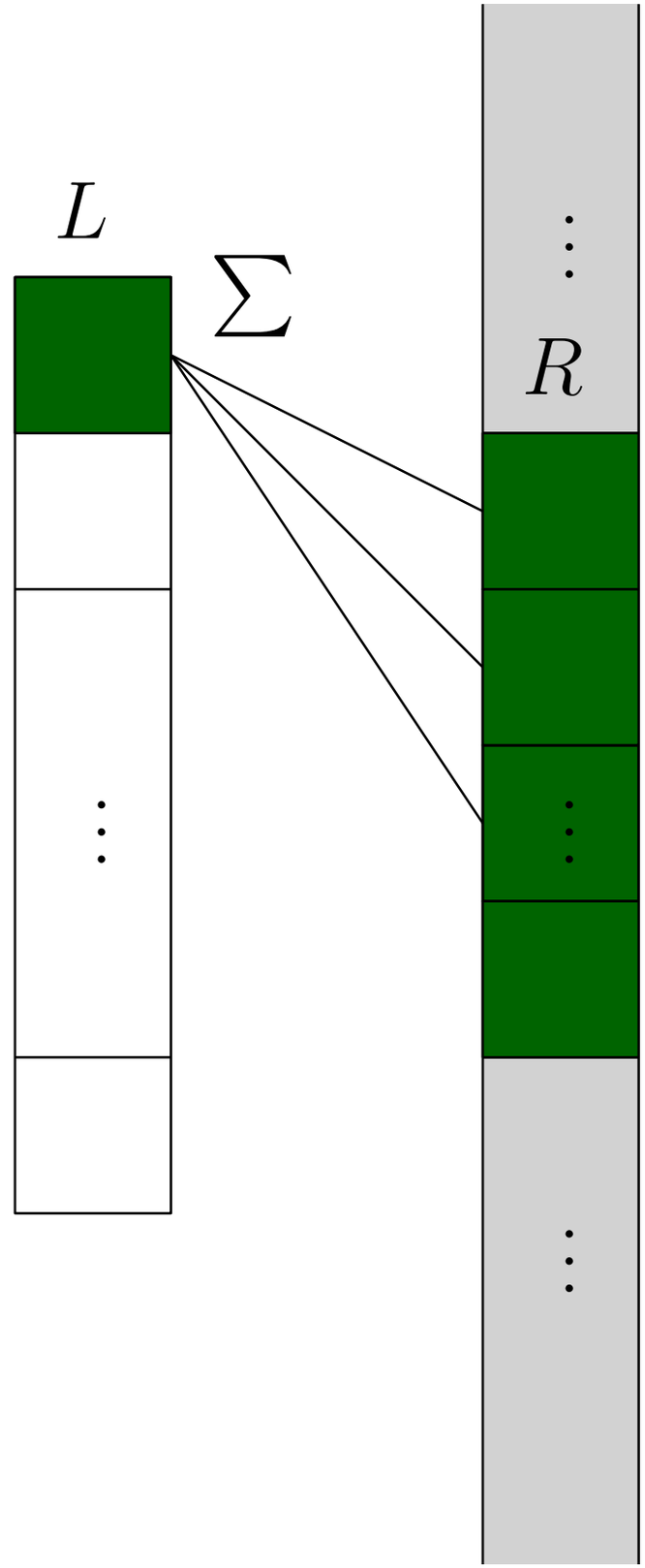
2. Expander hashing
   - Construct random Γ with:
     - Constant degree $d = 8$
     - $|L| = (\log^4 k)|R|$
   - Generation procedure

Example construction

Initialization and evaluation of local hash functions:

1. Multipoint evaluation
   - dk-independent polynomial hash function $h(x)$
     - Generate $|R| = \Omega(k \log k)$ values using time $O(k \log^4 k)$

2. Expander hashing
   - Construct random Γ with:
     - Constant degree $d = 8$
     - $|L| = (\log^4 k)|R|$
   - Generation procedure

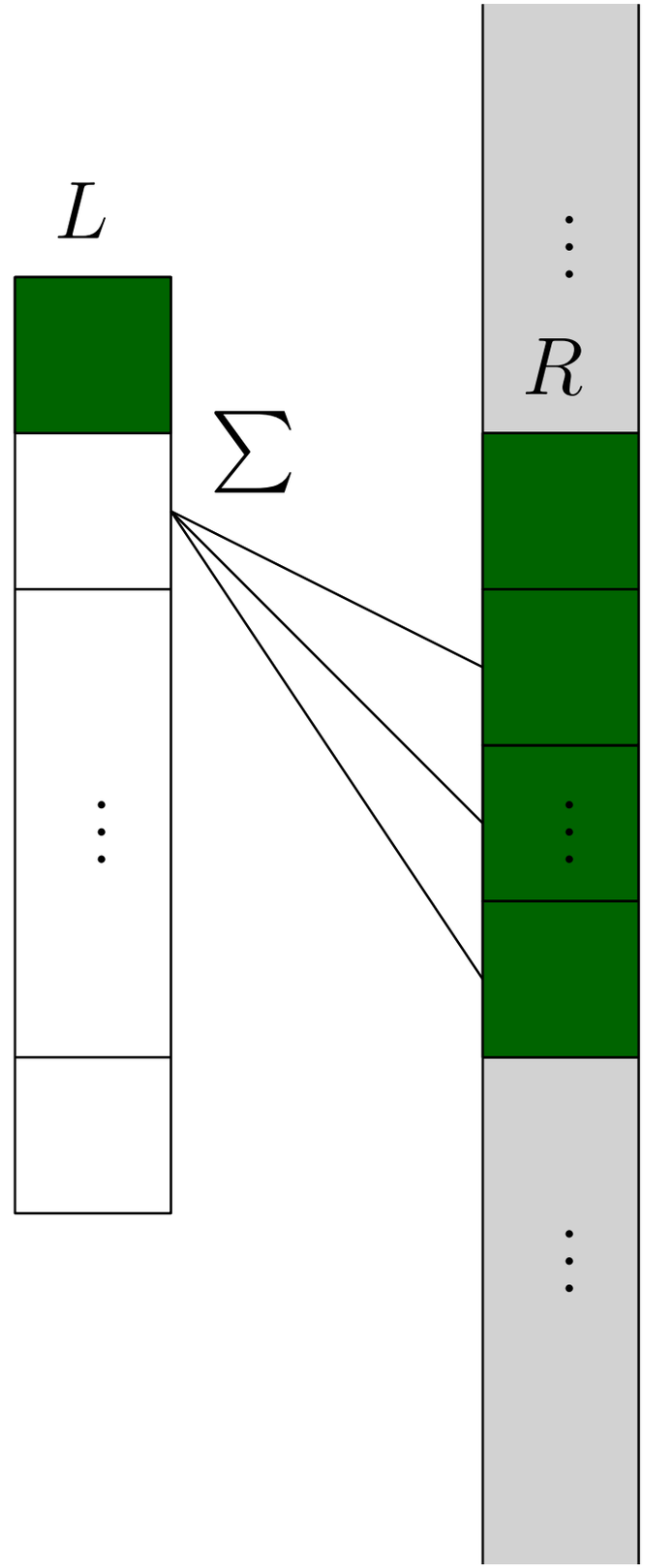


Example construction

Initialization and evaluation of local hash functions:

1. Multipoint evaluation
   - dk-independent polynomial hash function $h(x)$
     - Generate $|R| = \Omega(k \log k)$ values using time $O(k \log^4 k)$

2. Expander hashing
   - Construct random Γ with:
     - Constant degree $d = 8$
     - $|L| = (\log^4 k)|R|$
   - Generation procedure

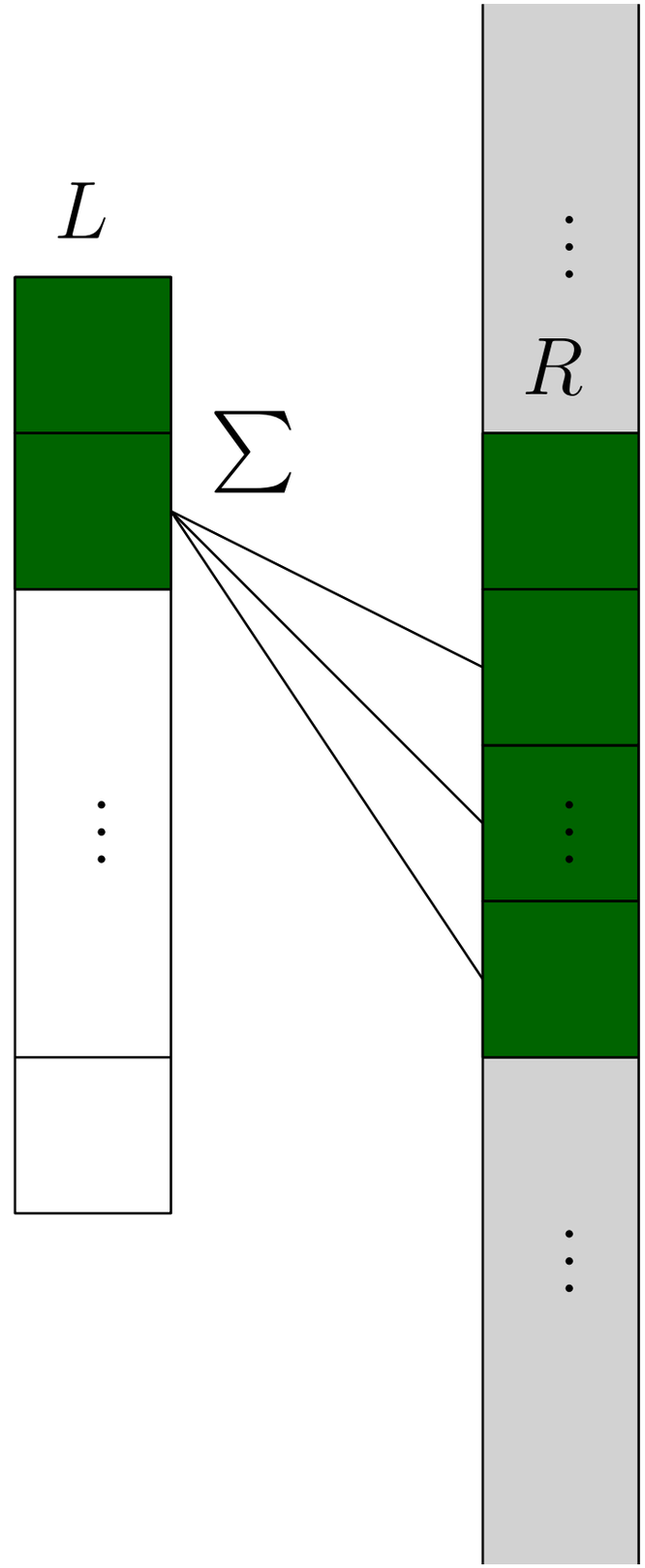


Example construction

Initialization and evaluation of local hash functions:

1. Multipoint evaluation
   - dk-independent polynomial hash function $h(x)$
     - Generate $|R| = \Omega(k \log k)$ values using time $O(k \log^4 k)$

2. Expander hashing
   - Construct random Γ with:
     - Constant degree $d = 8$
     - $|L| = (\log^4 k)|R|$
   - Generation procedure

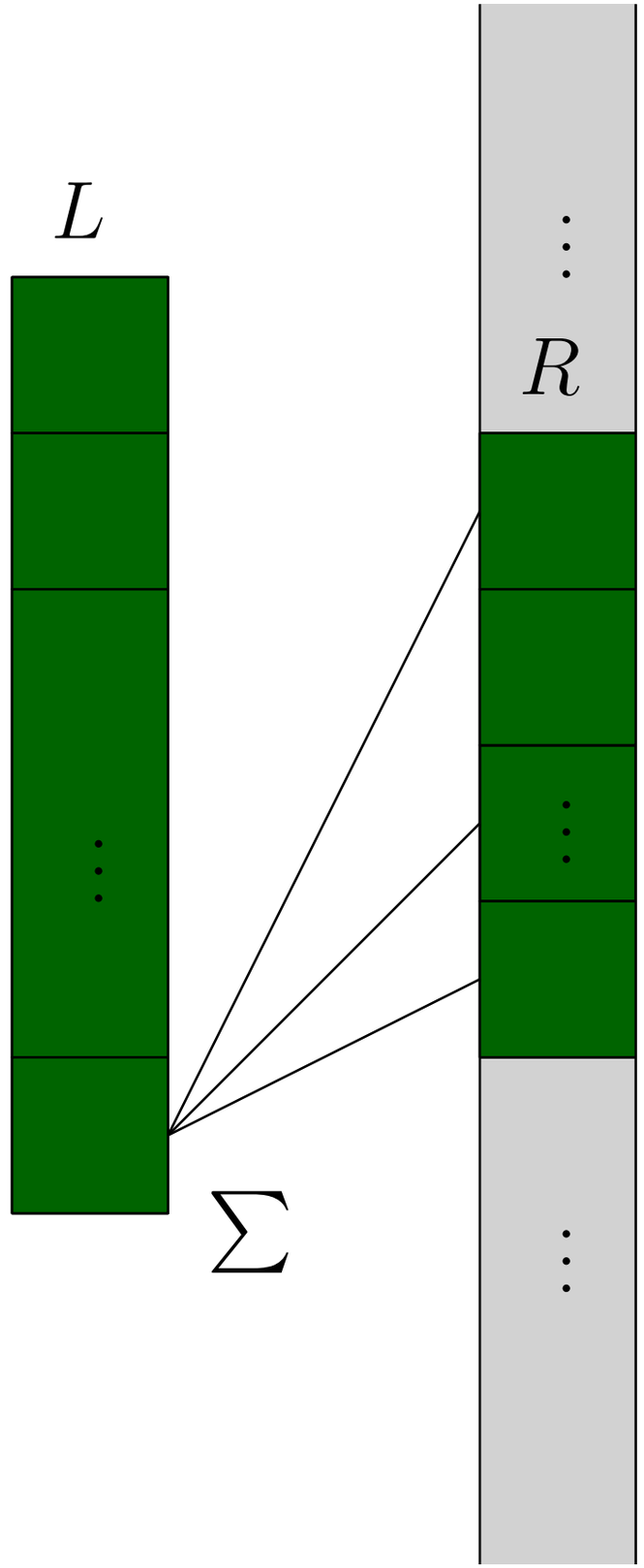


Example construction

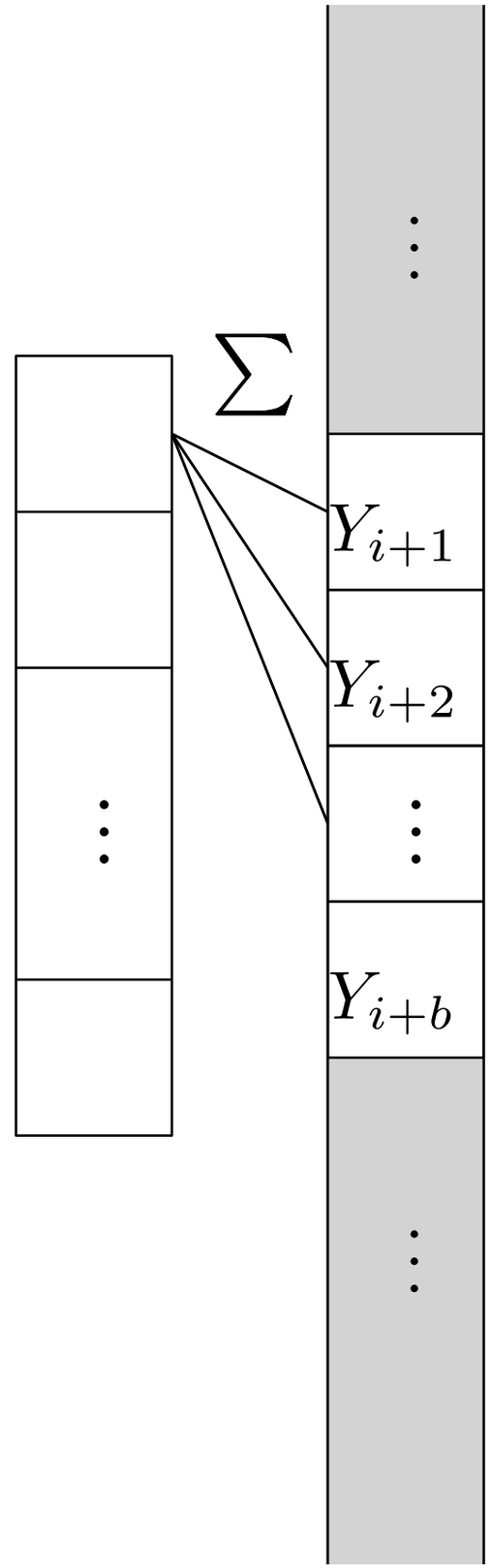
Initialization and evaluation of local hash functions:

1. Multipoint evaluation
   - dk-independent polynomial hash function $h(x)$
     - Generate $|R| = \Omega(k \log k)$ values using time $O(k \log^4 k)$

2. Expander hashing
   - Construct random Γ with:
     - Constant degree $d = 8$
     - $|L| = (\log^4 k)|R|$
   - Generation procedure



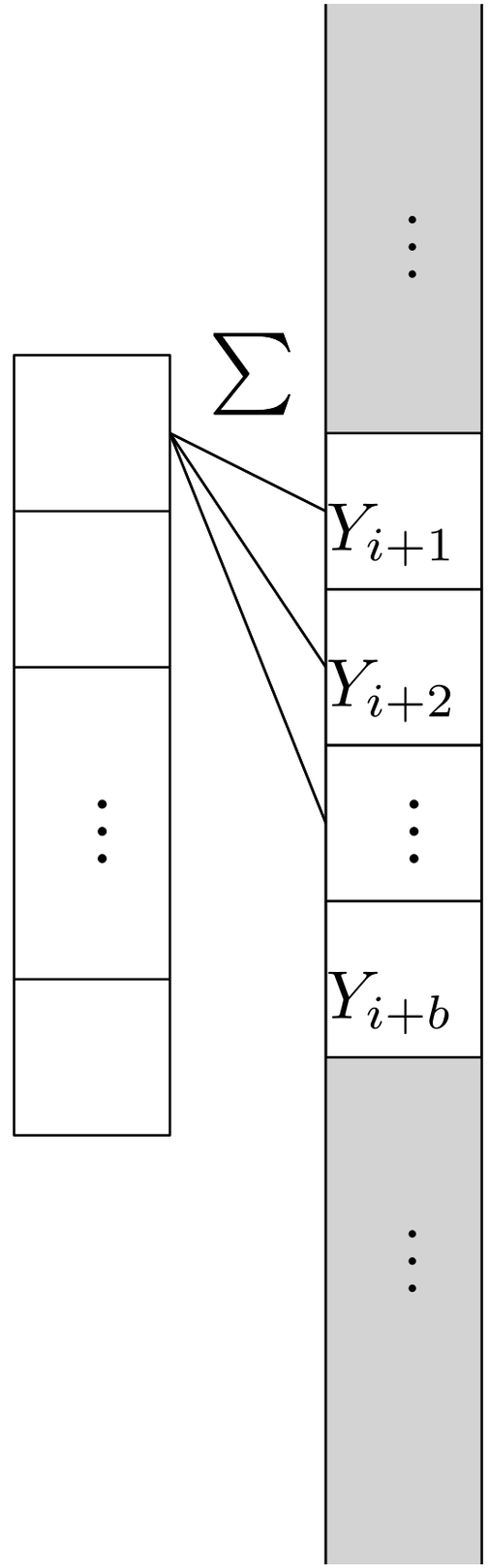
- No known explicit k-unique Γ with constant degree and $\operatorname{poly}\log k$ imbalance



Explicit k-generator

- No known explicit k-unique Γ with constant degree and $\operatorname{poly}\log k$ imbalance

- Explicit k-unique Γ [CRVW'02]
  - $|L| = c|R|$ for constant $c > 1$
  - $|R| = O(k)$
  - $d = O(1)$

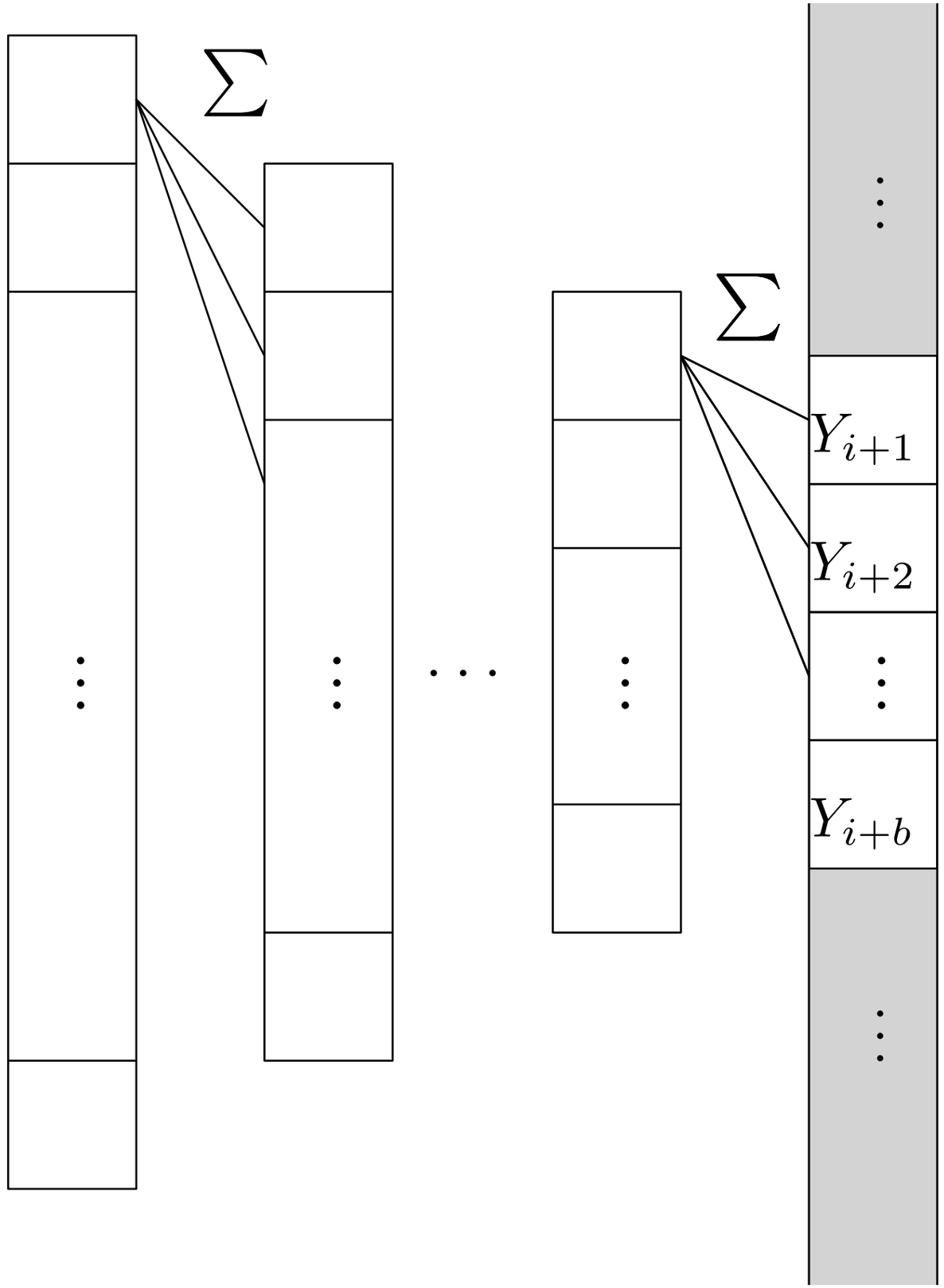


Explicit k-generator

- No known explicit k-unique Γ with constant degree and $\operatorname{poly}\log k$ imbalance

- Explicit k-unique Γ [CRVW'02]
  - $|L| = c|R|$ for constant $c > 1$
  - $|R| = O(k)$
  - $d = O(1)$

- Cascading composition of $O(\log\log k)$ expanders suffices for constant time generation



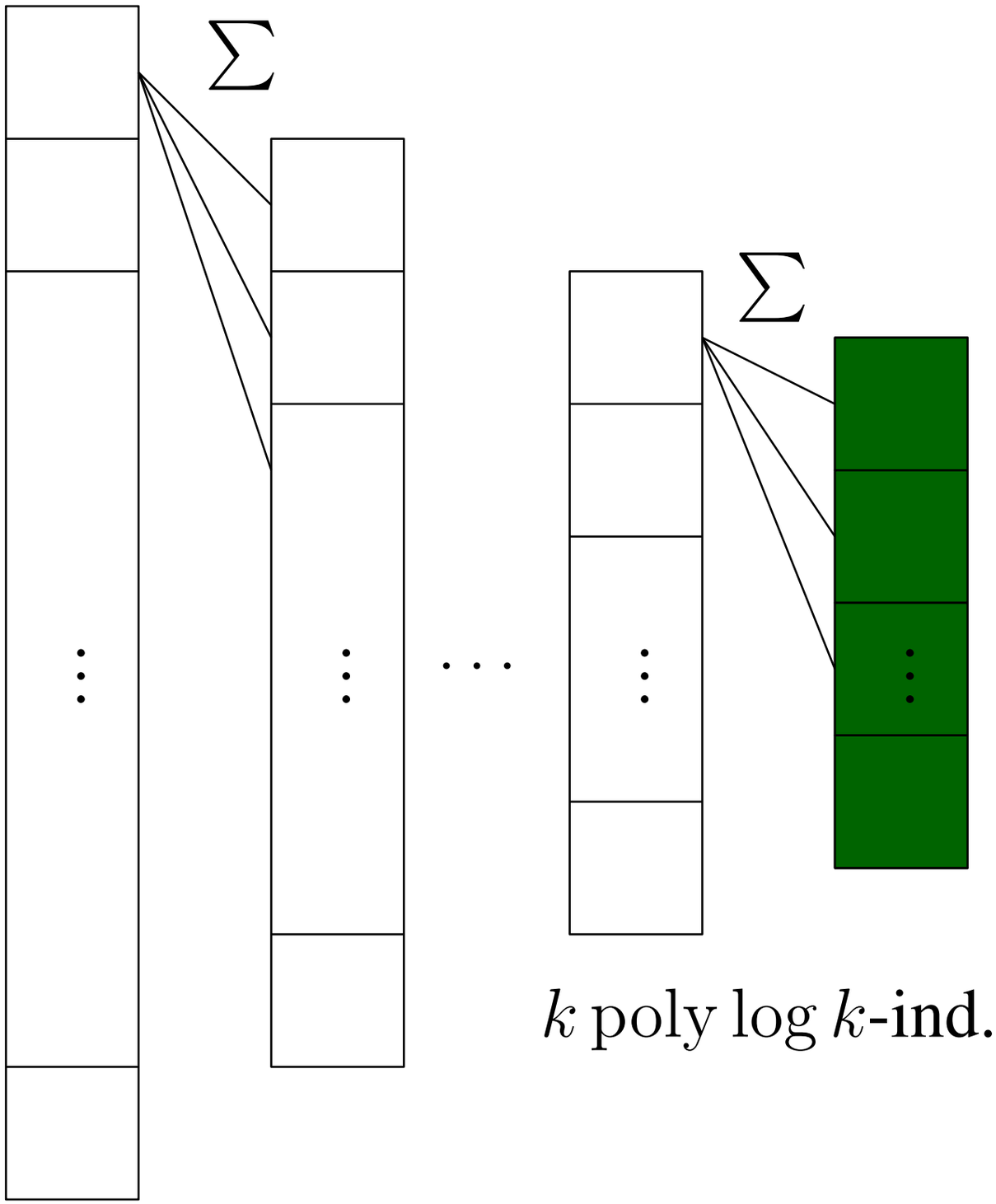
Explicit k-generator

- No known explicit k-unique Γ with constant degree and $\operatorname{poly}\log k$ imbalance

- Explicit k-unique Γ [CRVW'02]
  - $|L| = c|R|$ for constant $c > 1$
  - $|R| = O(k)$
  - $d = O(1)$

- Cascading composition of $O(\log\log k)$ expanders suffices for constant time generation

- Generation procedure



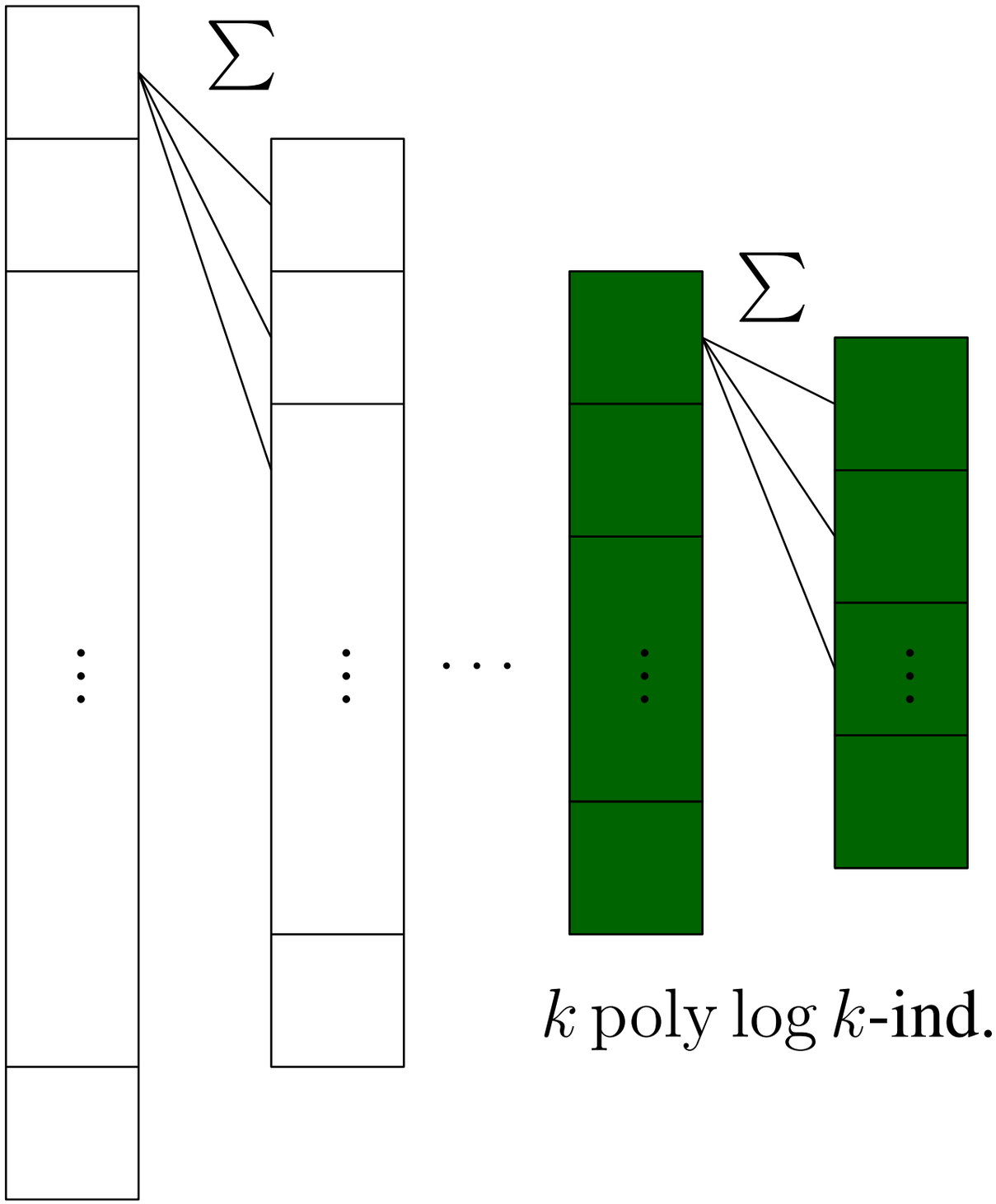
Explicit k-generator

- No known explicit k-unique Γ with constant degree and $\operatorname{poly}\log k$ imbalance

- Explicit k-unique Γ [CRVW'02]
  - $|L| = c|R|$ for constant $c > 1$
  - $|R| = O(k)$
  - $d = O(1)$

- Cascading composition of $O(\log\log k)$ expanders suffices for constant time generation

- Generation procedure



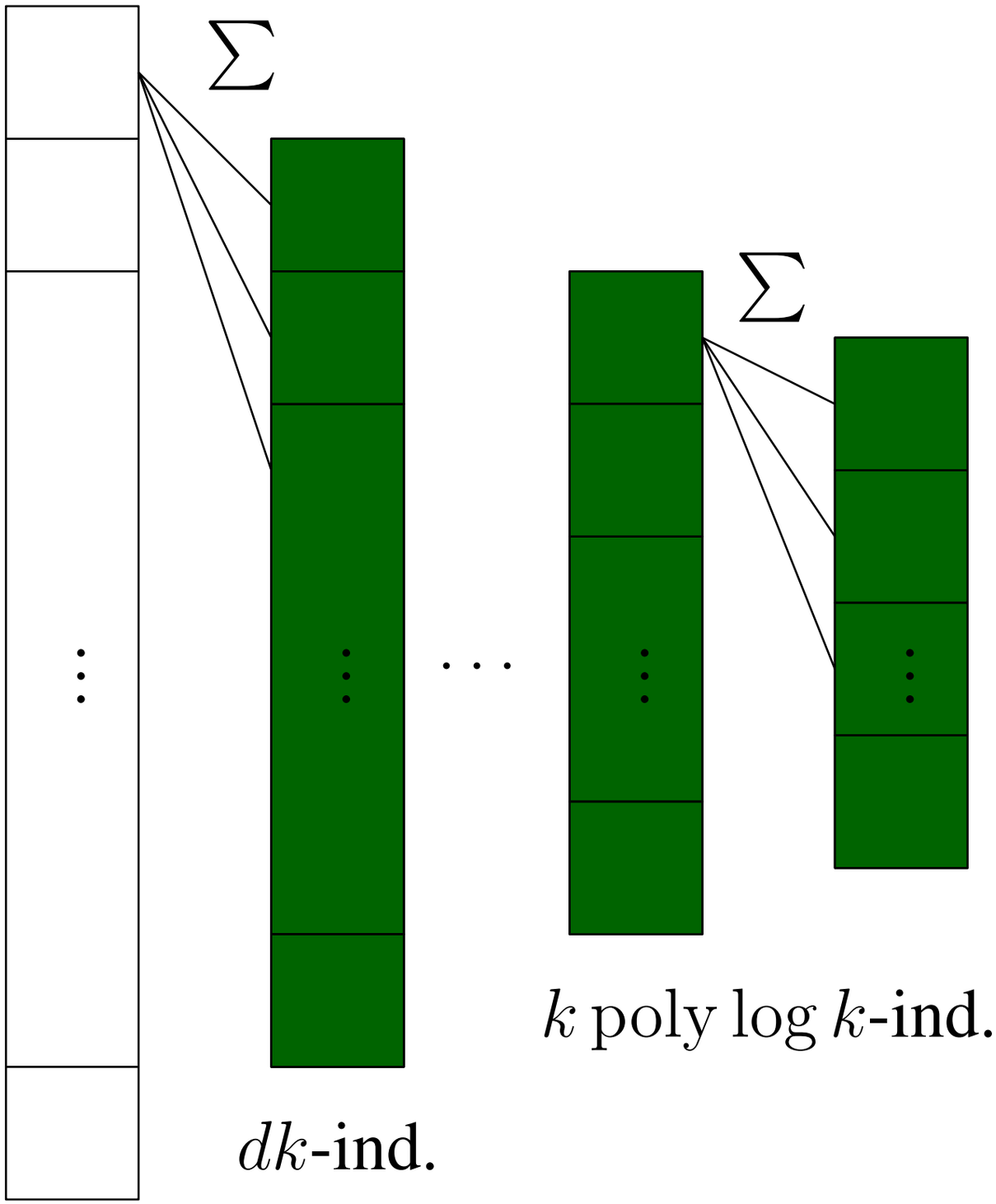
Explicit k-generator

- No known explicit k-unique Γ with constant degree and $\operatorname{poly}\log k$ imbalance

- Explicit k-unique Γ [CRVW'02]
  - $|L| = c|R|$ for constant $c > 1$
  - $|R| = O(k)$
  - $d = O(1)$

- Cascading composition of $O(\log\log k)$ expanders suffices for constant time generation

- Generation procedure



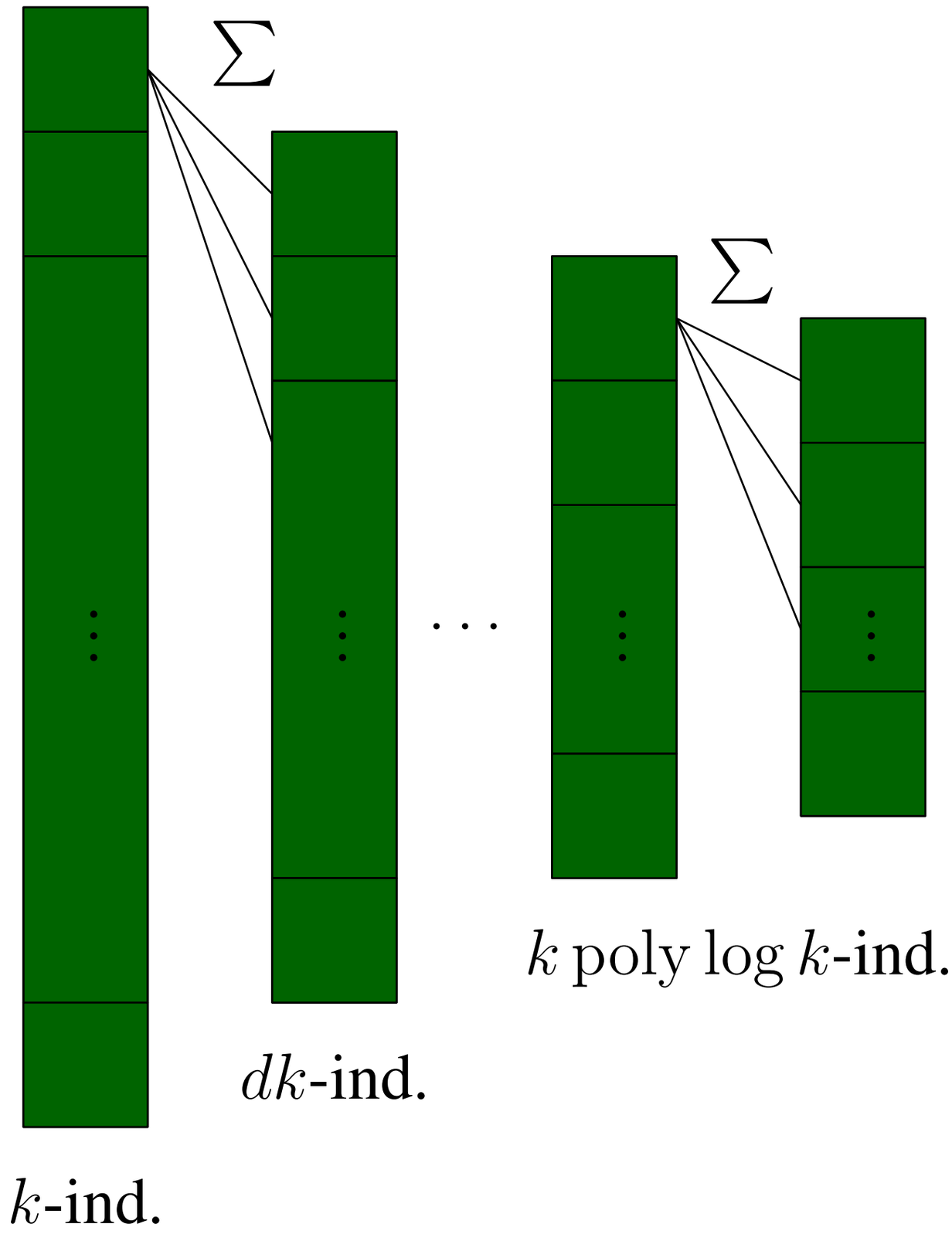
Explicit k-generator

- No known explicit k-unique Γ with constant degree and $\operatorname{poly}\log k$ imbalance

- Explicit k-unique Γ [CRVW'02]
  - $|L| = c|R|$ for constant $c > 1$
  - $|R| = O(k)$
  - $d = O(1)$

- Cascading composition of $O(\log\log k)$ expanders suffices for constant time generation

- Generation procedure



MILLIONS OF 64-BIT VALUES GENERATED PER SECOND, RANDOMIZED CONSTRUCTION

k	**Polynomial hash fct.**	**Multipoint [GM'10]**	**Our generator**
2^5	5.650	4.115	66.667
2^{10}	0.169	2.227	40.000
2^{15}	0.005	1.567	14.493
2^{20}	0.000	1.157	5.714

Experiments

MILLIONS OF 64-BIT VALUES GENERATED PER SECOND, RANDOMIZED CONSTRUCTION

k	**Polynomial hash fct.**	**Multipoint [GM'10]**	**Our generator**
2^5	5.650	4.115	66.667
2^{10}	0.169	2.227	40.000
2^{15}	0.005	1.567	14.493
2^{20}	0.000	1.157	5.714

- Millions of highly independent words per second
  - Potential application to Monte Carlo experiments

MILLIONS OF 64-BIT VALUES GENERATED PER SECOND, RANDOMIZED CONSTRUCTION

k	**Polynomial hash fct.**	**Multipoint [GM'10]**	**Our generator**
2^5	5.650	4.115	66.667
2^{10}	0.169	2.227	40.000
2^{15}	0.005	1.567	14.493
2^{20}	0.000	1.157	5.714

- Millions of highly independent words per second
  - Potential application to Monte Carlo experiments

- Gao-Mateer's additive FFT + Haswell support for $\mathbb{F}_{2^{64}}$
  - Evaluate a degree 2^{20} polynomial in 2^{20} points in < 1 second

Experiments

MILLIONS OF 64-BIT VALUES GENERATED PER SECOND, RANDOMIZED CONSTRUCTION

k	**Polynomial hash fct.**	**Multipoint [GM'10]**	**Our generator**
2^5	5.650	4.115	66.667
2^{10}	0.169	2.227	40.000
2^{15}	0.005	1.567	14.493
2^{20}	0.000	1.157	5.714

- Millions of highly independent words per second
  - Potential application to Monte Carlo experiments
- Gao-Mateer's additive FFT + Haswell support for $\mathbb{F}_{2^{64}}$
  - Evaluate a degree 2^{20} polynomial in 2^{20} points in < 1 second
- Slowdown of our generator as k increases is due to cache effects

Summary

Result:

- Explicit constant time generation using space $k \operatorname{poly} \log k$

Summary

Result:

- Explicit constant time generation using space $k \operatorname{poly} \log k$

Technique:

- Combination of multipoint evaluation and a cascading composition of explicit expanders

Summary

Result:

- Explicit constant time generation using space $k \operatorname{poly} \log k$

Technique:

- Combination of multipoint evaluation and a cascading composition of explicit expanders

Open questions:

- Explicit expanders with
  - $|R| = O(k \operatorname{poly} \log k)$
  - $|L| \geq (\operatorname{poly} \log k)|R|$
  - $d = O(1)$

Summary

Result:

- Explicit constant time generation using space $k \operatorname{poly} \log k$

Technique:

- Combination of multipoint evaluation and a cascading composition of explicit expanders

Open questions:

- Explicit expanders with
  - $|R| = O(k \operatorname{poly} \log k)$
  - $|L| \geq (\operatorname{poly} \log k)|R|$
  - $d = O(1)$
- Constant time generation with $O(k)$ space

Multipoint evaluation

- A k-independent polynomial hash function $h(x)$
- We can evaluate $h(x)$ in k arbitrary points $x_1, x_2, \ldots, x_k$ using $O(k \log^3 k)$ operations

$$h(x) = q(x)p(x) + r(x)$$

$$p_1(x) = \prod_{i=1}^{k/2}(x - x_i) \qquad p_2(x) = \prod_{i=(k/2)+1}^{k}(x - x_i)$$

$$r_1(x) = h(x) \bmod p_1(x) \qquad r_2(x) = h(x) \bmod p_2(x)$$

$$r_1(x_i) = h(x_i) \text{ for } 1 \leq i \leq k/2$$

- Faster algorithms exist if:
  - $x_1, x_2, \ldots, x_k$ lie in arithmetic/geometric progression
  - $\mathbb{F}$ contains a highly composite kth root of unity

New results

- We only require sequential evaluation of Γ
- Tabulating Γ requires at least $|R| = \Omega(k \log^{2+\varepsilon} k)$ words
- Constant time generator for k-unique $\Gamma : L \to R^d$ using space $O(k \log^{\varepsilon} k)$
- New recursive constructions of k-independent hash functions that come close to Siegel's lower bound